Password guessing attack based on PII

A password dictionary generation tool based on PI

Tazribine Ali

Research and Development project owner:
Tazribine Ali

Master thesis submitted under the supervision of
Debatty Thibault

in order to be awarded the Degree of
Master in Cybersecurity
Cryptalalysis and Forensics

Academic year
2023 – 2024

Password guessing attack based on PII: A password dictionary generation tool based on PII

Tazribine Ali: Tazribine Ali
Master in Cybersecurity – Cryptalalysis and Forensics
: 2023 – 2024

Abstract

In the dynamically changing domain of cybersecurity, password security continues to top the charts of the most controversial issues with ever-growing complexity in password-guessing attacks leveraging PII. This project addresses the challenge by proposing a new password generation tool that will work to enhance focused password guessing efforts. The goal is to enhance current tools like CUPP by using PII in password dictionary generation and utilizing probabilistic models to refine the guessing methodology.

The thesis is opened with a broad literature review of all existing authentication methods, pointing out the weaknesses of knowledge-based authentication when PII is used to construct passwords.

The study aims to design and implement a password generation tool, which was evaluated against CUPP with a series of metrics such as success rate, rank efficiency, and time performance. The evaluations were done on the Clixsense leaked dataset.

Experiments show that the new tool outperforms CUPP in terms of a higher success rate, fewer generated passwords during runtime, and more consistent ranking. These were ways in which this research established the power of targeted approaches driven by PII to inform password-guessing strategies. Significant limitations are also those constrained by the functionality of CUPP itself: the use of only one dataset and keeping dictionary sizes small due to computational restrictions. The thesis concludes with a discussion of the implications of these findings on the cybersecurity world, innovating in password security tools continuously, and articulating ways forward for future research, in particular, exploring machine learning techniques and testing on even more heterogeneous datasets. **Keywords:** PII, password guessing, dictionary, cupp, tool, generation

Table of Contents

List of Figures

List of Tables

List of Abbreviations

2FA	Two factor authentication
AAE	Adversarial AutoEncoder
AE	AutoEncoder
ASCII	American Standard Code for Information Interchange
ATM	Automated Teller Machines
DGM	Deep Generative Model
GAN	Generative Adversarial Network
GUI	Graphical User Interface
HFS	High Frequency Substring
LHS	Left-Hand-Side
MD5	Message Digest 5
MFA	Multi-Factor Authentication
NIST	National Institute of Standards and Technology
OTP	One Time Password
PCFG	Probabilistic Context-Free Grammar
PII	Personal Identifiable Information
PIN	Personal Identification Number
RHS	Right-Hand-Side
SFA	Single-Factor Authentication
URL	Uniform Resource Locator
VAE	Varitional AutoEncoder

Chapter 1

Introduction

In the digital age, password security assumes a place of pride in protecting personal and organizational information. Since cyberspace is dominated by online services ,today more than ever, strong passwords are very much needed. However, many users still stick to weak or easily guessable passwords, thus leaving their accounts open to attacks. Password guessing tools are used to test and improve password security. Such tools find weak passwords and give information to improve the policy on passwords.

1.1 Motivations

1.1.1 Context

Password security is the keystone of securing personal and organizational data. Today, with so much diffusion of the internet and digital services, everyone has dozens of online accounts that require strong passwords. Password-guessing tools were developed to exploit these weaknesses in user-created passwords. Early varieties of password-guessing tools relied upon simple brute force or dictionary attacks, wherein every possible combination or common password from a pre-defined list was tried for guessing. With improved security practices, new techniques were created.

Probably one of the most important developments in password guessing is the use of Personally Identifiable Information, PII. It contains personal details such as name, birthdate, and email addresses, all used to make the password guesses very accurate. Attackers exploit this PII because most users include personal information in their passwords for easy recall. To that end, incorporating PII into password-guessing tools can greatly improve chances of success.

Recent searches in cybersecurity revolve around understanding and mitigating risks associated with PII. With the increasing rate of data breaches, attackers have now access to vast volumes of PII, further increasing the threat. There is, therefore, a need for advanced tools capable of analyzing and exploiting PII in passwords to properly test the security implemented. Current trends also show a shift toward targeted attacks, in which the attacker makes use of peronal identifiable information (PII) to improve guessing techniques. This shift requires the development of state-of-the-art password-guessing tools which model such advanced attack methodologies.

1.1.2 Problem statement

Whereas password-guessing algorithms have undergone considerable improvement over this period, the existing tools that are PII-based, such as CUPP[1] (Common User Passwords Profiler) were not designed to combat today's security challenges. CUPP provides the foundation but without the sophistication needed to cope with both the complexities of current

[1] `https://github.com/Mebus/cupp`

password patterns and volumes of PII now readily available from data breaches. Current tools that are comprehensive in analyzing the usage of PII and its affecting passwords are very few, which is important to understand and mitigate this threat.

Advanced development of PII-based password guessing tools is not without challenges. Notably, one of the key problems that has to be addressed is incorporating large and varied PII datasets into the guessing algorithm. These tools must also adapt to the changing policies regarding passwords and behaviors expressed by users. Creating a fully functional PII analysis tool requires advanced data processing capabilities with effective identification and evaluation of the presence of a PII in a password.

The main research question this thesis is trying to answer is, "How can we build a more optimized PII-based password dictionary generation tool and a comprehensive PII analysis tool for passwords?" This question underscores how best to improve the accuracy and efficiency of PII-based guessing methods by including a robust tool in analyzing PII usage among passwords. An answer to this question would greatly benefit cybersecurity research in improving our capability to protect sensitive information from unwanted access.

1.2 Project statement & contributions

This thesis aims to build a PII-based password dictionary generation tool that will be better in performance than the already existing utility called CUPP. Although CUPP does formulate some basis regarding PII-based password guessing, it does not seem to be very effective. What we would want to do here is to construct something that exploits PII better with respect to the accuracy and efficiency of guessing passwords.

Further, we wish to implement a PII analysis tool for passwords. From our studies, we noticed a deficit in tools that could analyze the prevalence and impact of PII within passwords. This deep analysis tool will provide insights into how PII is used when users generate passwords and enhance security practice.

1.3 Organization of this document

This document has been designed to delve into some aspects of authentication, with a greater focus on password security, guessing techniques for passwords, and finally, the role of PII in the vulnerability of passwords. Then, it goes into the project statement and contributions.

Chapter 2 elaborates on the different mechanisms of authentication. The mechanisms range into three categories: knowledge-based, property-based, and biometric-based. This puts some foundation down for the authentication landscape.

Then, chapter 3 is entirely devoted to the subject of passwords, policies related to them, and implementation in the real world; it reviews human factors that impact password creation. Additionally, it explores some of the cognitive challenges users face, such as password reuse, memory problems, security vs. convenience, and common patterns and personal information impacting password security.

Chapter 4 explains, in detail, the techniques of password guessing, right from the traditional methods of brute force and dictionary attacks to probabilistic models and machine learning.

This chapter also reviews a variety of password cracking tools and defensive mechanisms so that one understands the practical difficulties and solutions before them in password security.

Chapter 5 discusses PII in password guessing, including a number of frameworks designed to exploit personal information for guessing passwords more effectively. This chapter underlines vulnerabilities stemming from the use of PII within the password.

Chapter 6 summarizes the mission, objectives, and requirements of the project. It explains precisely which specific goals are aimed at and within what scope of work undertaken. Moreover, it demarcates between requirements covered by the present state of the art or not, therefore setting a frame for the contributions of this thesis in the research context at large.

Chapter 7 presents the implementation and testing methodologies followed in the project. All tools and techniques used to achieve the objectives in the project are described here. This chapter also elaborates on the test strategy.

Chapter 8 describes how the setting of experimentation and the process of data collection ensues, with details on the environment in which hardware and software are used, along with tools and libraries, and some sources of data. The chapter explains in detail ways in which the experimental data has been prepared and processed.

Chapter 9 presents the results of experimentation and analysis targeting data from a leaked database. Detailed analysis has been done for password length, character makeup, presence of PII, and structural pattern characteristics in passwords. In addition to that, the implemented password generation tool has been compared to an existing PII based generation tool finally followed by a discussion of general findings.

Chapter 10 concludes the thesis by discussing the findings in relation to state-of-art, reflecting on the lessons learned, and identification of possible limitations of this work concerning the validity of results. Chapter 12 gives an indication of some directions for further research, and Chapter 13 gives a conclusions.

Chapter 2

Authentication

Authentication is very important when it comes to cybersecurity. It involves a check to verify that something or someone is what they claim to be. It is to ensure that the said information is accessed or tasks are performed by none other than the right people or systems. Sound authentication methods are crucial for safety and security in every system dealing with sensitive data.

There are three key types of authentication: knowledge-based, possession-based, and biometric-based. [20].

2.1 Knowledge-based authentication

The most important type of authentication for this thesis is knowledge-based authentication, which relies on user knowledge. This type is one of the most traditional forms of authentication and is widely used due to its simplicity and ease of implementation.

Generally, this type of authentication is separated into two types of methods: text-based authentication and image-based authentication.

In text-based authentication, the user has to answer a question with a secret that only the authenticated user knows. This secret could be a password, a PIN number, or even an answer to a security question like "What is the name of my first pet?".

An alternative to the above authentication type is a visual pass system, where users select images in a specific order on their GUI. Graphical passwords are classified using recognition-based and recall-based techniques. In recognition-based techniques, users are authenticated by recognizing images selected during registration. Recall-based techniques require users to reproduce what they generate during registration. [26]

2.2 Property-based authentication

Possession-based authentication is another type of authentication method that involves the use of something physical, such as a user's having to verify who they are. This is different from just using a password. For instance, when you are using an ATM, it requires the use of your bank card (something you have) and your PIN (something you know) to get money. There are two main types of these physical items, called tokens: hardware and software. Hardware tokens include USB security keys and smart cards, enabling them to be easily portable in daily routines. Software tokens enable the generation of OTPs (one-time passwords), which can be displayed on the user's phone or sent as an SMS. OTPs change in different parts of time, thus very hard to guess, leading to enhanced security [1, 21, 26].

In most cases, the possession-based mode of authentication is used in combination with other means, such as passwords or biometrics of fingerprints. This is what refers to multi-factor authentication. The more the methods, the more secure an account will be. Even if attackers manage to get the password, they will still need the physical token to get in [1]. Possession-based authentication is widely used in banks and big companies as they have to keep their information really secure. For example, a smart card used with a password may be immune to many kinds of attacks [26].

2.3 Biometric-based authentication

Biometric authentication is another method of using a person's physical or behavioral characteristics to identify them. Biometrics, unlike many other authentication methods, such as passwords that could be forgotten or sometimes stolen, relies on who you are, thus making it very strong in the context of security. It is characterized by the analysis of characteristics such as fingerprints, facial features, voice patterns, keystrokes, hand geometry, retina, and iris scans. The two general processes it goes through are enrollment and verification. In the stage of enrollment, the system captures several samples of a user's biometric trait, such as his fingerprint or the pattern of the iris, and stores the extracted features in a database. The next time the user makes a try at the system, the new sample under processing goes into the stage of verification to be compared for identity with the stored data [17].

One of the strengths of biometric authentication is the ability to provide evidence directly on who one is. Unlike passwords or PINs, which can be shared or stolen, biometric traits are unique and difficult to duplicate. This makes biometrics one of the technologies that enhance security in different applications. However, various biometric techniques offer different levels of accuracy and security. Fingerprint recognition is widely used, particularly in smartphones and laptops, mainly because of its convenience and effectiveness due to the high biometric availability of fingerprints. Nonetheless, it could be prone to spoofing since applicable are fingerprints lifted off from surfaces [19]. Iris and retinal scans are highly accurate options that require specialized, expensive equipment. These would include capturing and analyzing the unique patterns in the eyes and, therefore, be extremely reliable but not very common because of the cost.

The facial recognition system, on the other hand, has had vast improvements and is common on most consumer devices these days. It works by taking the measurement of the distance between key facial features, but it can be less reliable under different lighting conditions or with changes in the user's appearance. Voice recognition uses vocal traits for authentication but can be easily fooled by voice imitation or recordings. Hand geometry and vein recognition both measure the shape of the hand or its internal patterns, offering more reliable authentication, though they require specialized scanners. While biometric authentication has its advantages, it also has several challenges: This involves potential spoofing and the necessity for the protection of biometric data since the same cannot be changed once it has been stolen, unlike a password. It also raises privacy concerns since one's biometric data is very personal. In general, biometric authentication is a safe and handy means of identification, with its broad usage, especially when technology expands. However, more work in research is very necessary to address the security and privacy challenges in order to be adoptable by many more [17].

2.4 Summary

Authentication is important in cybersecurity because it ensures that sensitive information and systems are accessed or forwarded to authorized persons. There are three main classes of authentication methods in application: knowledge-based, possession-based, and biometric-based, all with varying strengths.

Authentication Type	**Description**
Knowledge-based	It relies on the information the user knows, like a password, PIN, or questions answers. It involves text-based techniques, such as passwords, and image-based techniques like graphical passwords. Although it is simple to implement, it is usually vulnerable to attacks like phishing or brute force.
Possession-based	Something the user has, such as a physical token or a device. Examples include bank cards and USB security keys. Possession-based authentication is often combined with knowledge-based ones to form multi-factor authentication (MFA). This makes the security stronger by requiring both a physical token and something the user knows.
Biometric-based	Leveraging unique physical or behavioral features, such as fingerprints, facial recognition, or iris scanning, to authenticate users. Biometric techniques have high security since it is hard to replicate them; however, this technique also has several privacy concerns and can be vulnerable to spoofing or data breaches.

Table 2.1: Summary of Authentication Methods

Chapter 3

Passwords

3.1 Introduction

Having learned about various authentication methods, we now focus on knowledge-based authentication, more precisely, passwords. As we have seen, information relating to personal identification has to be secured in the world of the Internet. This is when passwords come in. Passwords make sure that only a selected few can get hold of a particular piece of information: your email, your bank account, or even your social media profiles. Without strong passwords, other people might find it easier to access your accounts and rob your information.

A good password is like a solid lock on a door. It should be difficult for others to guess. This indicates that it should consist of some uppercase letters, lowercase letters, numbers, and even special symbols, such as @ or #, and should be as long as possible. For example, the length of one 12-character password is much stronger than a 6-character password. However, people still tend to create simple passwords and maintain the same password for different accounts. That is substantially dangerous. For a hacker who has acquired one password, access to many accounts is easy. Recent studies underline the need for unique and complex passwords for every account [5].

This is where a password manager might come in handy. A password manager creates strong passwords and remembers them for you so that you only remember one master password. This way, having different strong passwords for each account is quite effortless [2]. Having said that, another way to protect the account is by going through two-factor authentication (2FA) or multi-factor authentication (MFA). This technique would allow you to provide two pieces of information or more to log in, for example, a password and a code sent to your phone. The effectiveness of such a measure increases because, even if the hacker knows your password, he still needs the second code to access your account [20].

Hackers use various methods to steal your passwords. One way is phishing, in which they will trick you into giving up your password by appearing to be a trusted source, such as your bank. The other manner is the utilization of software that makes trial guesses of a large number of diverse passwords extremely quickly until it gets it right; it is referred to as a brute-force attack. And there are other kinds of attacks. Studies indicate that upon learning more about these threats, users can remain protected [23].

3.2 Password policies

To ensure the production of secure passwords, The National Institute of Standards and Technology (NIST) has set out full guidelines [21]. The suggestions on creating and remembering and the required security procedures for passwords will ensure that a user is well safeguarded from potential risks. If followed, these regulations would enable an organization or an individual to improve their security regarding password use, thus minimizing unauthorized breaches.

Password Complexity and Length

NIST suggests that passwords should be at least eight characters long, but longer passwords are recommended. NIST no longer requires or recommends a mix of uppercase letters, lowercase letters, numbers, and special characters but advises allowing any type of character and spaces to allow for passphrases that are more easily remembered and typed.

Password Creation

NIST does not support the use of complex composition rules or require the use of different characters because this tends to produce predictable patterns. Instead, they suggest using password blocklists to stop common, expected, or compromised passwords. Users shall be allowed to create passwords up to 64 characters in length to encourage passphrases.

Password Expiration

NIST advises that organizations do not impose periodic password changes except when a breach has been detected. Frequent forced changes can result in poor password choices since users will tend to make minimal modifications to existing passwords.

Password Entry

NIST says that masking (showing asterisks) during password entry should be optional. Users may have a preference to see the actual characters entered to allow for reduced error. Moreover, the rate of incorrect password trials should be controlled to avoid brute-force attacks.

Multi-Factor Authentication (MFA)

NIST recommends that multi-factor authentication should be applied when possible. MFA is a process in which something you know is combined with something you have (like a password or security token) or something you are (like biometric verification).

Password Handling

Passwords should be stored securely using salted cryptographic hashing algorithms. Insecure storage of passwords in plain text is strongly discouraged. Besides, it is important to have passwords transmitted across an open network secured so that they are not intercepted.

3.3 Reality of password policies

While NIST suggests clear guidelines available on its website, in practice, the password policies set on most websites can deviate from these. The new guidelines issued by NIST in 2017 have been shown to exhibit discrepancies between guidelines and reality recently by Sahin et al. [24]. This study draws attention to the strategies of implementation as well as the policy formulation made by web admins regarding setting a password policy for their websites. It reflects stricter policies so that a snapshot of how usual password policies are currently placed on real websites is obtained.

Sahin et al. [24] also point out that most admins are still using conventional password policies. Newer recommendations suggest longer passwords instead of complex ones and even recommend not changing them periodically. Yet, many administrators still revert to making the passwords complex and requiring their users to change them every once in a while. All these have their roots in older guidelines, such as the 2004 NIST recommendations, indicating much work is still needed to adopt the new standards.

The study also indicates that in most cases, administrators are still unaware of many best practices at the time of research, such as advising that the password is not changed frequently. Mostly, password policy decisions are made by heads or management of the organization, who tend to have outdated ideas about password security. This highlights the need for more information and awareness so that administrators can understand and implement the guidelines provided.

The current documents on the standards are too long, tiring, and complex; thus, it becomes a real challenge for the administrators to execute and interpret the documents. Sahin et al. [24] recommended that if guidelines are provided through a summary table with proper explanations, this will assist administrators in understanding and applying such recommended policies.

3.4 Human behavior in password creation

Password choice is a very important aspect of online security, and a majority of people find it difficult to observe best practices in this area. Knowledge of human behaviors in such an area would be quite essential in developing better measures for security.

3.4.1 Password reuse

Password reuse is the habit of using the same password for more than one purpose. Actually, this is very common because people currently handle a lot of accounts. Research shows that many people would reuse passwords not because they intend to but rather because it is easier than remembering a different password for every account [29].

Despite making life easier, reusing passwords creates drastic security risks. Basically, if one password is divulged through a website, then access can be gained to other websites where the same password has been used. In this manner, there seems to be a domino effect where, through one breach, many accounts can be accessed. For example, if one has the same password for social media, banking, or email, then any single breach in these services compromises the integrity of all the individual user's accounts. [33].

3.4.2 Cognitive load and memory challenges

The notion of cognitive load is relevant here. Cognitive load relates to the effort of mental capacity needed to process and retain information. For the user, passwords would lead to a high cognitive load since they require memorizing many unique and complex passwords. Modern requirements on passwords to contain a mixture of both case letters, numbers, and even special characters further aggravate the situation [29].

Human memory is limited. It is difficult for many humans to remember a multiple number of complicated passwords. They reuse passwords, therefore, to make their online experience simpler. Wash, and Rader [33]. Reported that the users often consider easiness and memorability rather than security in most cases, which leads to reusing their passwords.

3.4.3 Simplicity and remembrance capability

Another important aspect of password selection is simplicity and remembrance capability. Even though users know that simple passwords are less secure, they commonly tend to choose simple passwords that can be remembered easily. This is because of cognitive load and the need to remember many passwords [29].

3.4.4 Common patterns and personal information

People use common patterns for information such as number or letter sequences and personal information such as names and birth dates, among others. While these would make a password relatively easy to remember, they make it equally easy for an attacker to guess [29].

3.4.5 Common words and phrases

Another strategy for making passwords memorable is to use common words or phrases. However, this makes the passwords weak because dictionary attacks can be carried out by an attacker. Knowing the risks, users usually choose convenience over security [29].

3.4.6 Impact of password policies

Strict password policies are enforced with the aim of making passwords hard to guess. Sometimes, however, the whole process may backfire. Instead of learning the complex conditions required, users usually resort to ways of coping, like adding a number to a familiar word or using predictable patterns. These may, in a way, meet the policy requirements but really do not make passwords much more secure at all [13].

3.4.7 User frustration and non-Compliance

Complex password policies can also lead to user frustration. Writing down passwords or using the same ones on multiple sites are major threats to security, and people will take the easiest route. So, policies need to strike a balance where the best choices are also easiest, and security fatigue is avoided [13].

3.4.8 Security/Convenience tradeoff

Essentially, good password management is about a tradeoff between security and convenience. Balancing strong and secure passwords against those that are easy to remember poses a challenge for users. The tradeoff in the development of strong but memorable passwords is influenced by cognitive shortcuts, fear of forgetting, and perceived risk. Users tend to prefer convenience over more secure and less convenient passwords [29].

3.4.9 Security awareness and its impact on password choices

It is important for users to know of security threats that could enable them to appreciate the significance of having strong passwords. However, awareness does not generally translate to better behavior. Most users exhibit cognitive dissonance, where beliefs about needing strong passwords do not match the actions of those who select weaker ones. Practical solutions, for example, the use of password managers, can easily and without much cognitive load manage several complex passwords [33].

3.5 Summary

Key Learning	Summary
Importance of strong passwords	They are very important in securing digital accounts and keeping unauthorized access at bay. This is what will repel any forms of security break-ins at the very frontier.
Effective Password Policies	It is the length, not complexity, that password policies should pursue and they encourage long, memorable passphrases. Mandatory frequent changes to passwords are not encouraged since they may result in weaker passwords..
Challenges in Implementing Password Policies	Still too many organizations hold old practices for the enforcement of unnecessary complexity and frequent change that can result in quite predictable patterns of passwords, reducing security overall.
Human Behavior and Password Creation	A lot of pitfalls make this rather challenging, such as cognitive load, memory limitations, and a tendency to reuse. All of these factors result in the generation of simpler, less secure passwords.
Security Awareness and User Education	awareness of security risks do not always result in better practice. There should be practical tools, such as for instance password managers and multi-factor authentication, to guide secure password habits..

Table 3.1: Summary of Password chapter

Chapter 4

Password Guessing

4.1 Introduction

Arguably, password guessing is the most common and one of the oldest threats in the whole cybersecurity landscape. This technique involves revealing the password by systematically checking a large number of possible variants. Attackers have many password-guessing methods that exploit weak and predictable password choices, user behavior, and system vulnerabilities. Understanding the basics of password guessing is essential for both implementing effective defenses and educating users on the importance of strong password practices.

Password guessing attacks can range from simple measures of guessing common passwords to sophisticated strategies that use advanced algorithms with large datasets. The complexity and uniqueness of the passwords and the security measures in place to mitigate unauthorized access determine the effectiveness of these attacks.

One basic concept that governs password guessing lies in the predictability of human behavior as seen in **section 2.4.4**. Most users prefer simple passwords that they can remember easily, using basic combinations of words, numbers, and common phrases. Being so predictable, it is easier for attackers to guess passwords when users use the same password across different accounts.

To protect against password guessing, strong policies should be enforced, and extensive user education programs should be carried out regarding creating complex and unique passwords. In addition, the use of multi-factor authentication and other advanced security features can significantly reduce the risk of successful password-guessing attacks.

This topic defines types of password-guessing attacks with basic and advanced features. Further in the chapter, we will review popular software tools with functionality applied in practice. Protection approaches from these attacks are also discussed to understand all the pieces of the system that define a security strategy to mitigate risks of harm from the method of guessing passwords.

4.2 Types of password guessing attack

4.2.1 Online vs offline password guessing

Online and offline methods are the two ways in which password-guessing attacks can be divided, and they each have their own unique difficulties and tactics. It is important to recognize these categories so as to develop effective defenses.

A method of guessing a user's password online requires an attacker to make multiple login attempts directly on the system. This technique presumes that the individual attempting it will have to communicate with a computer until he or she gets the right combination of symbols, letters, numbers, etc. [31].

The number of guesses allowed before intervention by systems is one major shortcoming of online password guessing. A majority of systems employ security mechanisms such as account lockout policies or even IP address blacklisting to limit unsuccessful tries. For instance, if the login attempts are not successful after several tries, then there would be a temporary blockage or permanent one on the account. The process also slows down due to network latency, thereby limiting the speed of attack [31, 32].

Despite these limitations, online password guessing is still employed by attackers who exploit target information like frequently used passwords, user IDs, birth dates, or other personal details. Most often, they start with the most probable passwords because it is understood that many users prefer easy and foreseeable ones [31, 32].

Offline password guessing occurs when a hacker has access to hashed password data from a leaked database or compromised system. Unlike online attacks, offline password guessing does not require continuous interaction with the targeted system, which allows an intruder to work independently and at his pace [31, 35].

One major advantage of offline attacks is that there are no limitations on the number of guesses. The attacker can attempt several million passwords without detection or being restricted by system policies. This is because they operate with hashed values of passwords and employ powerful machines to calculate hash results faster than any online environment would be able to manage [31, 32, 35].

Offline password guessing typically involves the use of advanced algorithms and techniques such as dictionary attacks, brute-force attacks, PCFGs (probabilistic context-free grammars), and Markov models. These methods are used to create possible passwords using different sources of data such as common password lists, leaked password databases, patterns of keyboard use, and linguistic patterns [31, 35].

Offline attacks, however, pose a serious threat once access to hashed password files can be achieved. To enhance password security, systems should, therefore, make use of a strong hashing algorithm plus salting technique [31, 35].

So, even though both offline and online guessing carries considerable risks, they have differences in terms of execution and limitations. Online attacks are restricted by network speeds and system defenses, but offline ones do not suffer from this problem, thus allowing an unlimited number of guesses without detection. It is crucial to understand these variations in order to build effective defense mechanisms against both kinds of attacks that will provide complete protection from any possibility of password breaches.

4.2.2 Targeted vs trawling password guessing

Password guessing attacks can be broadly categorized into two distinct types: targeted and trawling attacks. Each type uses different strategies and focuses, making it crucial to understand their differences to develop effective defenses.

When a hacker specifically wants to break into a particular user's account, he employs a targeted password-guessing attack. Such attacks are employed by the attacker when he or she has a well-defined target in mind, such as crawling into an online account or restoring a user's lost password. This technique involves collecting as much information as possible about the target so as to reduce the search space [31].

Attackers often use publicly available personal information from social media or other online sources. By leveraging this data, they can create highly probable password guesses that match the target's habits and preferences. This personalized approach makes targeted attacks particularly dangerous, as the guesses are not random but informed by specific user details [35].

In contrast, trawling password-guessing attacks aim to crack passwords across a large group of users. This type of attack does not target specific individuals but seeks to find any user accounts that match a known password. Trawling attacks are often used in scenarios where password storage or hashed values have been compromised. The primary goal is to uncover as many passwords as possible, making it a broad and sweeping approach [31].

Unlike targeted attacks, trawling does not rely on personal information. Instead, it uses common passwords, dictionaries of leaked passwords, and patterns observed in large datasets. This method is efficient in identifying weak passwords that are frequently used by many users. The effectiveness of trawling attacks is enhanced when attackers have access to large collections of hashed passwords, allowing them to run their algorithms offline without the limitations imposed by online systems [35].

One key difference between these two types of attacks lies in their mode of operation. Targeted attacks are typically performed online, requiring the attacker to interact with the system in real time, submit the user identity, and then try the guessed passwords. This means the number of attempts is often limited by the system's security settings, such as account lockouts after a certain number of failed attempts [31].

On the other hand, trawling attacks are generally performed offline. Here, the attacker works with hashed password values and can make an unlimited number of guesses, constrained only by computational resources. This offline approach allows attackers to use powerful hardware and sophisticated algorithms to guess as many passwords as possible, making it a potent method for compromising large numbers of accounts simultaneously [35].

4.2.3 Brute Force Attacks

Brute force attacks are one of the most straightforward yet powerful techniques used in password cracking. The idea is very simple: just try all possible combinations until you get to the password. This approach will definitely yield the password provided that adequate time and processing power are supplied, though at very high values for time and resources. However, brute-force attacks can be very time-consuming and resource-intensive.

All possible combinations are tried one after another in a brute-force attack until the correct password is identified. Now, if the password is eight characters long and contains only lowercase letters, there will be 26^8 different combinations. That would be 208827064.576 possibilities! Even if there is the likelihood of the attacker's computer running through 1000 passwords per second, all possible combinations would require more than 208 million seconds, so approximately 58000 hours. Clearly, brute force attacks are not quick and often not feasible for long and complex passwords [22, 34].

It becomes really complex when dealing with encrypted passwords. Many systems keep passwords hashed by algorithms like MD5. In such a situation, to make the attack successful, the attacker has to generate the hash for every possible password and then match it with the stored hash. For example, some early versions of Linux kept passwords in MD5 hash format. If an attacker steals the file containing these hashed passwords, they could use brute force to compare the generated hashes until a match is found [22, 34].

Even though they are simple, brute force attacks remain a big threat, largely to short and less complex passwords. In most cases, attackers will always target systems where they can leak the hashed passwords to perform an offline brute force attack without them being detected. This is due to the fact that offline attacks are not strengthened by the security features imposed by a system, such as a lockout after a certain number of attempts on account login [22].

Although one of the most basic and primitive ways to crack a password, brute force attacks are very dangerous. Knowing how they work might help users and system administrators implement appropriate security measures that would greatly reduce the risk of successful brute-force attacks.

4.2.4 Dictionary Attacks

Password guessing is a well-known method that hackers use, and it is known as a dictionary attack. However, unlike brute force attacks that attempt every possible combination of characters, hackers use pre-made lists of likely passwords making them generally much quicker and more effective.

In a dictionary attack, the attacker uses a collection of words commonly used as passwords. These words often come from everyday life, like names of birds, places, or famous people. Users often choose these kinds of passwords because they are easy to remember [22]. However, many password policies require modifications to these words to make them more secure, such as adding numbers or special characters.

To get around this, attackers apply "word-mangling" rules to the words in their dictionary. These rules systematically alter the words to create variations that users might use. For example, a common rule might add a number to the end of a word, so "password" becomes "password9". Another rule might change the first letter to uppercase, turning "password" into "Password." By applying these rules, attackers can generate a large number of possible passwords from a relatively small list of dictionary words [34].

Choosing the right word-mangling rules is crucial for the success of a dictionary attack. Each rule can produce many variations of a single word, and using multiple rules in combination can exponentially increase the number of guesses. For example, if there is an 800,000-word dictionary and two digits are added at the end of each word, this will generate 80 million guesses while changing only the first letter to upper and lower cases doubles it up. Therefore, attackers need to select rules that maximize their chances of success while keeping the number of guesses manageable [34].

While dictionary attacks are faster than brute force attacks because they don't have to check every possible combination, they still have limitations. The biggest limitation is that the dictionary might not contain the password being guessed. If the user has chosen a password that is not in the dictionary, the attack will fail. Nonetheless, these kinds of attacks are very fruitful since many users select simple common passwords that an attacker would probably have in his or her dictionary [22].

In summary, dictionary attacks are a faster and often more efficient method than brute force attacks for guessing passwords. By using common words and applying systematic variations, attackers can generate a wide range of possible passwords. But then again, the effectiveness of this attack mainly depends on how good the dictionary itself is and what rules have been applied for mangling words. Possessing knowledge about these attacks can assist in creating better preventive measures against them.

4.2.5 Hybrid Attacks [12, 18]

Hybrid attacks combine both dictionary and brute force attacks into one very effective method. These attacks start with a list of widely used passwords (a dictionary) and then apply systematic variations to enhance the success rate. Combining the two methods helps attackers guess slightly modified common words or phrases for passwords much faster than using just one method alone.

A hybrid attack is based on a base dictionary of likely passwords, to which various modifications are applied, similar to those made by users to meet complexity requirements. These include adding numbers or special characters to the beginning or end of a word, substituting characters with look-alike symbols, or using multiple words. For example, "password" can be changed to "Password1," "P@ssword," or even "123password."

The strength of hybrid attacks lies in their ability to cover a broad range of potential passwords without actually trying every possible combination. By focusing on typical patterns and predictable variations, these attacks can easily break passwords based on simple modifications of dictionary words. This method is most effective against passwords that users believe are secure because they contain some additional characters but are based on recognizable patterns.

Hybrid attacks exploit the predictability of human behavior in creating passwords, making them very effective in the hands of an attacker. Most users create their passwords from known words with minor modifications, believing this will be enough to be secure. On the other hand, hybrid attacks take advantage of these predictable patterns to guess passwords more easily.

For example, a user might start with a known word such as "password" from the dictionary and then apply a rule to add a number at the end, resulting in guesses like "password1," "password2," and so on. One might also use common substitutions, such as replacing "a" with "@," resulting in "p@ssword." Another strategy is combining words like "password123admin."

4.2.6 Rainbow Table Attacks [3, 7]

Rainbow table attacks are a form of attack against cryptographic hash functions wherein huge communities of precomputed values are passed to reverse hash functions, very often more effectively than in standard rainbow attacks. So, if one user chooses a password, it gets hashed by a cryptographic hash function and then stored in a database. This is mapped back by rainbow tables to its plaintext passwords, bypassing the generation and comparison of hashes on the fly by attackers.

To construct a rainbow table, attackers apply a reduction function that will take a hash and revert it back to a plaintext guess. Chains of plaintext-hash pairs are built from this data and stored in the table. At attack time, the attacker looks up the target hash in the table and traces through the chain to recover the password.

The rainbow table attack is way faster than the brute force attack because it makes use of pre-computed data. Another drawback of these tables is that they require a huge amount of storage and computational resources for generation. They then become quite poorly performing under robust hashing algorithms with salting mechanisms in their design—for instance, adding a unique value to each password before hashing and eventually producing two different hashes from similar passwords.

Rainbow table attacks can be limited to a substantial extent by using robust hash functions with good salting. Enforce multi-factor authentication and train the end-users to change their passwords frequently. The complex password policies can help in preventing such an attack to a huge extent.

4.2.7 Phishing Attacks

Phishing is a web-based attack whereby the attacker misleads the user into giving out sensitive information like passwords or PIN codes through redirection to similar-looking websites. For example, when one inputs "www.yahoo.com" with an intent to visit, they may actually be directed to "www.yah0o.com," which is identically made to look like the latter. Thus, under the impression that it is real, one will enter the login details grasped by the attacker. After gathering the login credentials, the attacker may redirect the user to the real website, logging him or her in without the user noticing that anything has happened [22].

The phishing attacks, therefore, cash in on the trust users have in familiar websites, thereby making it hard for any vigilant user to differentiate between the two. Though various phishing filters and controls are put in place, they cannot be entirely trusted since a majority of the phishing sites end up bypassing these defenses. This form of attack is very hazardous since such an attack could result in high-security breaches as the attackers get personal and financial information.

Knowledge of phishing and how it works is quite important in enhancing cybersecurity awareness and the implementation of better protective means. Always validate the URL, be suspicious if unsolicited requests for personal information are made, and use security software that aids in the detection of such phishing attempts. User education on indicators of phishing, combined with careful online behavior, is a critical defense against this very common threat.

4.2.8 Shoulder Surfing

Shoulder surfing is a very basic yet effective technique where the attacker observes the user while he/she enters their password. The attacker can acquire the information directly by watching the victim or with the aid of tools. Despite the simplicity of this attack technique, it has turned out to be quite effective due to the human tendency to ignore physical security.

The most straightforward form of shoulder surfing involves an attacker who gets close enough to the target to view both the keyboard and monitor while the victim is inputting their password. Crowded places like cafes, airports, or even offices may provide such chances. However, it is in no way limited to direct observation. Attackers can implement many methods to spy on targets from a distance. For instance, they may use binoculars to observe from a distance because by doing so, they can remain unnoticeable themselves yet obtain the required information [22].

More sophisticated methods include the use of hidden cameras, such as CCTV, to record what a victim does. Cameras can be strategically placed to get good views of the keyboard and the monitor from very advantageous angles. Sometimes, attackers even use audio cues. By the number of keystrokes, they could determine the length of the password and then try all possible combinations of that many characters in length, as shown by Raza et al. [22].

Although it doesn't need a lot of technology, the threat posed by shoulder surfing can be quite significant. This involves hanging about in public or at least semi-public settings where sensitive information is frequently accessed. Users must, therefore, become all the more aware of their surroundings, taking precautions like shielding keyboards when typing and becoming conscious of who might be watching. There are also screen privacy filters that blur a monitor from side views to a degree where an attacker may struggle to get information.

4.2.9 Summary

Attack Type	**Description**
Online vs Offline Password Guessing	Whereas the online attack requires several login attempts directly to a system, it is usually very limited due to defenses added within the system, such as account lockout. The offline attacks will happen if the hashed passwords have been attained, making it possible for an attacker to make limitless guesses outside the system constraint.
Targeted vs Trawling Attacks	Targeted attacks are when somebody is particularly targeting a few persons, mostly playing with personal information to make educated guesses, and trawling attacks aim to compromise any account by using common passwords across a large user base.
Brute Force Attacks	This technique works on the logic wherein all possible combinations of passwords are tried until the right one is found. Clearly, this could end up being very time-consuming and resource-intensive for long passwords
Dictionary Attacks	Uses rainbow tables of common passwords and a list of "word-mangling" rules to generate all the possible variations. Much faster than a brute-force attack but is still limited by the comprehensiveness of the dictionary.
Hybrid Attacks	This combines dictionary and brute-force attacks, starting with common words and applying systematic variations, thus being effective against slightly modified passwords.
Rainbow Table Attacks	This type of attack involves using precomputed tables that hold hash values to reverse engineer hashed passwords. Effectively used against unsalted hashes, it is mitigated with salting techniques.
Phishing Attacks	Fooling users to get their passwords through fake websites or emails.
Shoulder Surfing	A physical attack where the attacker looks over your shoulder as you type in your password. Simple but effective under the right conditions.

Table 4.1: Summary of Password Guessing Attack Types

4.3 Advanced Password Guessing Techniques

4.3.1 Probabilistic Context-Free Grammar (PCFG)

Probabilistic Context-Free Grammar (PCFG) is an advanced technique used in password cracking that leverages the structure and patterns found in human-generated passwords. This method prioritizes the generation of password guesses based on their likelihood derived from previously observed password data.

PCFG makes use of a process in which the generation of passwords is entirely modeled based on grammar. Characterizing variables are defined for the different types of characters, like letters or digits, together with their arrangement rules to form a password. The real key to the strategy of PCFG is the assignment of probabilities to these rules in accordance with their frequency in a training set of existing passwords.

Training phase: The first step in using a PCFG-based password cracker is the training phase. In this phase, a large set of previously disclosed passwords is analyzed for common patterns and structures. Such passwords are parsed down into what is called a base structure, which is simply a sequence of character types. For example, "P@ssw0rd123" would be parsed into the structure $L_1S_1L_3D_1L_2D_3$ where L = letters, S = special characters, and D = digits.

Every base structure in the training set is followed by its corresponding frequency. All this frequency data comes into the calculation of probabilities of different structures. Besides, probabilities of certain digits and special characters in passwords have been deduced. For example, a base structure like $S_1L_8D_3$ could have occurred with a probability of 0.1 in the training set. The second type of information obtained from the training set was the probability of digit strings and special strings appearing in the training set.

Using context-free grammar: For the longest time, context-free grammar has been part of any natural language designed to obtain strings with specified structures through generation or parsing. PCFG uses this same approach for guessing passwords that are similar in form to those created by humans. As explained by Weir et al. [34] a context-free grammar may be defined as $G = (V, \Sigma, S, P)$, where:

- V is a finite set of variables (or non-terminals)
- Σ is a finite set of terminals
- S is the start variable
- P is a finite set of productions of the form $\alpha \to \beta$ where α is a single variable and β is a string consisting of variables or terminals

Probabilistic context-free grammars assign a probability to each production, in such a way that for any particular left-hand-side variable, all the productions with this variable as the LHS add up to 1. Given these training sets, PCFG produces one set of productions that generates base structures and another set that generates terminals comprising digits and special characters. An example of such a set of production that generates base structures is as follows:

$$S \to L_1D_1S_1 \quad (P = 0.50)$$
$$S \to L_3D_1S_1 \quad (P = 0.30)$$
$$S \to L_2D_1S_1 \quad (P = 0.20)$$

An example of such a set of production that generates terminals is as follows:

$$D_1 \to 1 \quad (P = 0.50)$$
$$D_1 \to 2 \quad (P = 0.25)$$
$$D_1 \to 3 \quad (P = 0.25)$$
$$S_1 \to ! \quad (P = 0.40)$$
$$S_1 \to @ \quad (P = 0.30)$$
$$S_1 \to \# \quad (P = 0.30)$$

The probability of a sentential form, which is a string derived from the start symbol, is the product of the probabilities of the productions used in its derivation. For example, with the grammar $S \rightarrow L_3D_1S_1 \rightarrow L_32S_1 \rightarrow L_32\#$, the pre-terminal $L_32\#$ will have probability $0.3 * 0.25 * 0.3 = 0.0225$.

The grammars we derive are unambiguous context-free grammars. Given a pre-terminal structure, a dictionary is used to derive a terminal structure, and this is your password guess. For example, if you have a dictionary that contains {cat, hat, stuff, monkey}, the pre-terminal structure $L_3$4! would generate the following two guesses, the terminal structures: {cat4! hat4!} since those are the only two dictionary words of length three.

4.3.2 Markov Models [31]

Markov Models are a major password-cracking technique and were popular, especially before the inception of deep learning. More specifically, Markov Models gained such great popularity in the natural language processing and text area because they are conditioned to predict the order of characters based on probability relations. They are used to guess the following characters in a password, given patterns of characters used by real users to formulate passwords, and they perform very well in creating probable password permutations.

The main idea behind Markov-based password guessing is to use the relationships between adjacent characters in a password to predict the next character. This concept is based on the observation that letters in human-generated passwords are not chosen independently but follow certain regular patterns. For example, in English, the sequence "th" is much more common than "tq," and the letter "e" is likely to follow "th."

In a Markov Model, the probability of a password of length m (denoted as $c_1c_2\dots c_m$) is estimated using the formula:

$$P(c_1c_2\dots c_m) = P(c_1c_2\dots c_n)\prod_{i=n+1}^{m} P(c_i|c_{i-n+1}\dots c_i),$$

Where n defines the number of history characters used to predict the next character. This is also referred to as the order of the model. The combination of n history characters and the predicted character forms an $(n+1)$-gram fragment. For instance, in a 2-gram (bigram) model, the next character is predicted based on the previous character.

Before using Markov Models to guess passwords, we need to train it on a real password dataset to infer the initial probabilities, $P(c_1c_2\dots c_n)$, and transition probabilities, $P(c_i|c_{i-n+1}\dots c_i)$, with respect to a given order n, in order to use them for password guessing. Actually, this will cut each password in the training set into $(n+1)$-gram fragments. All the (n + 1)-gram fragments in the training dataset are sorted with respect to their conditional characters, that is, the first n characters of each fragment. Then, for every group, the probability of each (n + 1)-gram is calculated.

Suppose an example of training a 2-order Markov Model with the following passwords in the training dataset: "password," "passphrase," and "passcode." Sure, the train breaks these passwords down into bigrams (2 grams) like "pa," "as," "us," etc. Then, it will determine transition probabilities, for instance, the likelihood that "a" follows after "p" or that "s" succeeds "a." These probabilities are used to predict the next character when generating new password guesses.

For example,

- The probability of "a" given "p" might be high if many passwords in the training set contain "pa.".
- Similarly, "ss" might frequently follow "as."

These probabilities are then used by the Markov Model in the generation of probable password guesses by predicting the next character with respect to the previous characters.

4.3.3 Machine Learning-Based Attacks [31]

Machine Learning methods have improved password-guessing strategies. The techniques leverage high-end algorithms that enable the prediction of passwords more effectively than traditional methods. In this section, an overview is presented of how different Machine Learning techniques have been applied in password guessing.

Recurrent Neural Network (RNN)

Recurrent Neural Networks are specialized for sequential data, as in time series signals and natural language. This processing step by sequence can be done one at a time, with the output from one influencing the next. However, it has been shown that standard RNNs do not work too well on very long sequences due to the vanishing gradient problem. Long Short-Term Memory Networks were created to resolve this problem. LSTMs are basically a type of RNN designed to handle long-term dependencies. The first LSTM-based password model was presented by Melicher et al. [16]. This model uses a 3-layer LSTM to predict the next character of a password based on previous characters. For example, for the password "pass12", it first predicts the probability of 'p.' Using this 'p," it predicts 'a'a, and so on, until the whole password is predicted. One-hot encoding is applied, where each character gets converted into a numerical vector, and the characters correspond to unique vectors. This does quite a good job of predicting the probability distribution of every subsequent character in a password.

Encoder-Decoder Architectures

The Encoder-Decoder architecture is another approach that captures how people reuse passwords. The pass2path model is one such example which uses a 2-layer LSTM with residual connections. For example, this architecture predicts transformation paths such as substituting, inserting, or deleting characters to form new passwords. That is, the encoder maps the input password into a latent vector, whereas the decoder generates transformation sequences to produce new password candidates. This method emphasizes password distribution and transformation rather than direct prediction.

Deep Generative Models

DGMs learn the distribution of training data to generate new passwords. Among them, two common approaches are Generative Adversarial Networks and Autoencoders.

- Generative Adversarial Networks (GANs): Proposed by Ian Goodfellow [8], GANs consist of a generator and a discriminator. The generator generates the password samples that are, in nature, fake, while a discriminator is trained to distinguish between the real and fake samples. In the process of this adversarial training, the generator learns to generate very realistic passwords.

- AutoEncoders (AE): These map the input data to a latent space and reconstruct it from there. Examples include variational AEs (VAEs) and adversarial AEs (AAEs), where the latent space follows some distribution that allows one to sample new data. They can learn complex distributions over passwords and generate a variety of high-quality password guesses.

Hybrid models

Hybrid approaches combine several methods for guessing passwords. For instance, we can combine probabilistic context-free grammars with BiLSTM (Bidirectional LSTM) networks. In this approach, PCFG is used to create the base structure of the passwords, and then these structures train a BiLSTM network. This will enable us to use all the powerful features of grammar and neural network approaches in one approach since it returns structurally correct, contextually accurate passwords.

4.3.4 Summary

Technique	**Description**
Probabilistic Context-Free Grammar (PCFG)	Generates password guesses using a probabilistic model following linguistic patterns and structures from known passwords. Effective at real password guessing.
Markov Models	This technique makes use of Markov chains in guessing probable character sequences involved in passwords and is therefore based on some statistical analysis of password datasets.
Machine Learning-Based Attacks	This makes use of machine learning algorithms that learn patterns from large datasets of passwords, hence improving the efficiency and success rate of these guessing attacks.

Table 4.2: Summary of Advanced Password Guessing Techniques

4.4 Password cracking tools

A good number of tools and software have been developed to help with password cracking and security analysis processes in cybersecurity. These range from very simple scripting tools targeting brute force attacks to very complex programs that make use of advanced techniques such as John the Ripper, Hydra, and Hashcat.

4.4.1 John The Ripper

John the Ripper[1] is a very popular and powerful open-source password-cracking tool. Most importantly, it is applied in the recovery of passwords from hash values, and so is an offline password-cracking tool. Most security professionals and ethical hackers use it to test the strength of passwords and to validate how many weak passwords are assigned in a system. John the Ripper has many types of password-cracking attacks built in, which include dictionary attacks, brute force attacks, and hybrid attacks. It works against a great variety of hash types and encryption algorithms, such as DES, MD5, SHA-1, SHA-256, SHA-512, and many more. The user can configure this tool according to his requirements. The tool supports custom rules and expands with extra applicable modules and plugins.

John the Ripper is performance-optimized. Indeed, it uses multiple CPUs and even hardware accelerations like GPUs to increase performance in the cracking process. There is a large, active community supporting the development and upkeep of this tool. There's also a commercial version called John the Ripper Pro, which has a lot of extra features and optimizations.

The very first step in using John the Ripper is to obtain the password hashes that are to be cracked. For instance, the /etc/shadow file for UNIX-based systems contains hashes. Users choose the crack mode that is appropriate for them.

The most common modes are:

- Single Crack: the info from the username and related, etc., is used in guessing passwords.
- Wordlist Mode: One uses a dictionary list for common passwords with rules to create variations.
- Incremental Mode: It tries all permutations of characters (brute-force).

The selected mode and options will start processing the given hashes by running John the Ripper, which will keep trying different passwords one after another until it finds corresponding ones or runs out of possibilities. Once you have finished pausing the attack, check your cracked passwords, and accordingly, you can take necessary action, like informing users to change weak passwords or to make the system more secure.

Ethical hackers use John the Ripper to test password strength in a system and find weak passwords. System administrators use it to check password policies and enforce stronger password requirements. In digital forensics, it can be used to recover passwords from seized devices or data. John the Ripper is a very useful and powerful tool for password cracking and security testing. Its support in hash types, cracking methods, and customization makes it invaluable for security professionals in their mission to better systems.

[1]`https://www.openwall.com/john/`

4.4.2 Hashcat

Hashcat[2] is another very popular and powerful offline password-cracking tool that has been designed specifically for high-performance cracking. It's very fast, flexible, and able to harness a number of hardware accelerations. HashCat is optimized for better performance and can utilize the power of GPUs, CPUs, and other hardware accelerators. This makes it considerably faster compared to many password-cracking tools.

Hashcat supports a lot of hash algorithms, such as MD5, SHA-1, SHA-256, SHA-512, NTLM, WPA/WPA2, bcrypt, Kerberos, and many more. It also supports several attack modes designed for different reasons of cracking. It could use:

- Dictionary Attack: runs a preset list of possible passwords
- Brute Force Attack: takes all possible combinations of characters
- Mask Attack: similar to brute force but with more control over the pattern of characters
- Rule-based Attacks, which apply rules in modifying entries in a dictionary
- Combination Attack: two or more dictionaries combine words
- Hybrid Attack: combination of dictionary attacks and brute force

It allows users to granularly configure the tool with respect to certain rules, masks, attack strategies, etc., so as to have an optimum cracking process.

Hashcat works on Windows, Linux, and macOS. It is open source; thus, its development and improvement are open to the community. To get started in using Hashcat, you must first gather the hashes you want to crack. These can come from files or databases, among others. You select the appropriate attack mode depending on the scenario. You define parameters such as the setup of an attack through the definition of a hash type, dictionary file, mask, or rules. Then, you successfully run Hashcat with the set options, and it will efficiently total an attack using available hardware. You then look through the cracked passwords and perform relevant actions for security enforcement.

4.4.3 Hydra

Hydra[3] is a very efficient and widely applied password-cracking tool; this tool is majorly aimed at network protocol brute force attacks unlike tools like John the Ripper and Hashcat, which are primarily used for offline password cracking, Hydra has been designed to attack online services targeting several network services in a search for valid login credentials.

Hydra comprises many network protocols and thus can be used to test security in various services. Some of the supported protocols include HTTP, HTTPS, FTP, SSH, Telnet, SMTP, IMAP, POP3, SMB, RDP, MySQL, PostgreSQL, Oracle, LDAP, VNC, SNMP, and many others. Hydra attempts logins parallel, so brute-forcing against sequential attempts is very fast. It allows users to set several options and parameters that adapt the attack according to their needs by using usernames, passwords, wordlists, and other protocol-specific options. Hydra is modular. That means it is relatively easy to add new protocols

[2] `https://hashcat.net/hashcat/`
[3] `https://salsa.debian.org/pkg-security-team/hydra`

and improve existing functionality. It can either be interactive or run in batch mode, so it's flexible for use-case scenarios.

First of all, to use Hydra, you have to identify the target service and the protocol that you're going to test. This can be everything from a web login form to an SSH service to an FTP server.

Gather information; this may further include possible usernames and passwords. Either predefined wordlists or self-made lists can be used for this purpose. Set a target, protocol, and other options; list username and password in the Hydra command. After doing that, run Hydra with the set options. It should be noted that Hydra will try to log into the service using the provided credentials. Finally, go through the results and see what are the valid login credentials. If successful, take appropriate actions to secure the service and prevent unauthorized access.

4.4.4 Medusa

Medusa[4] is a robust and fast parallel password cracker. It's mainly used for online brute-force attacks against various network protocols and services. In most cases, it plays the same role as Hydra, although with a few minor differences. Medusa offers support for a large number of network protocols and services, including FTP, HTTP, IMAP, MSSQL, MySQL, NCP, POP3, PostgreSQL, RDP, SMB, SMTP, SSH, Telnet, VNC, and many others. This means that Medusa is able to perform independent login attempts simultaneously, thus speeding up the brute force process.

Medusa has been designed to be modular, so new modules for extra protocols can be added straightforwardly. This design finally enables users to change attacks according to their needs.

The options Medusa provides range from specifying username and password lists down to specific user–password pairs, with the addition of delay between attempts and other parameters controlling the process of brute force. It also provides advanced logging and output options for tracking the progress of any attacks and thus analyzing their results effectively. To use Medusa, first, the target service and protocol to be tested must be identified. You can either use the already available large wordlists or create your own. Finally, specify the target and protocol with other target options like username and password lists in the Medusa command. Finally, run Medusa with the options configured.

Medusa will attempt to log in to a target service in parallel with the given credentials. After that, analyze the results for valid login credentials. If successful, do what is necessary to secure the service and avoid further access by unauthorized entities.

4.4.5 Summary

[4]`https://salsa.debian.org/pkg-security-team/medusa`

Tool	**Description**
John The Ripper	One of the most popular open-source password-cracking tools, designed for brute-force and dictionary attacks, supporting a vast array of password hashes.
Hashcat	Very good at password cracking, uses GPUs to enable fast hashing and supports many different algorithms like MD5, SHA-1, etc.
Hydra	Very fast network logon cracker, which supports many different protocols as well as performs dictionary or brute-force attacks against remote authentication services.
Medusa	This is similar to Hydra, but designed for speed and large-scale brute-force attacks against remote authentication systems.

Table 4.3: Summary of Popular Password Cracking Tools

4.5 Password dictionary generator tools

To perform a dictionary attack, we need a good dictionary to enhance the chances of guessing the password. A password dictionary generator tool helps in the creation of comprehensive and effective word lists targeting specific targets. This includes common passwords, phrases, and patterns that users would use, hence increasing the chance of success in an attack.

4.5.1 Crunch

Crunch[5] is a wordlist generator program that is run from the command line and became very famous due to its usage in password cracking. This tool is very flexible and able to produce a wordlist based on criteria specified by the user, like character sets, lengths, and patterns.

The tool supports custom character sets, which means users can specify which characters are to be used in generating the wordlist; this could be letters, numbers, and special characters. In addition, it generates variable lengths that can go from a minimum length to a maximum of user choice in creating passwords of specific lengths.

Then, Crunch allows pattern-based creation. Users can create patterns that can tell how the generated passwords should look, including the fixed and variable characters. Moreover, it is possible to save this wordlist into a file or pipe it to other tools directly, like John the Ripper or Hashcat.

Finally, Crunch can compress the output through gzip. This reduces the storage requirements of large-word lists.

Here's a step-by-step process on how to use crunch:

1. character set definition: This defines what characters to use in generating passwords. These can comprise letters, either lower or upper-case, numbers, or even special characters.

[5] `https://github.com/jim3ma/crunch`

2. Set the length: This is where you specify the minimum and the maximum password length for which one is to be generated.
3. Create pattern: Build up patterns as desired to structure the passwords one way or another. You could, for example, fix some of the characters at particular positions or maybe vary them within a given set.
4. Generate wordlist: Just simply run the Crunch command prior to specifying all options. This will have Crunch generate the wordlist, output it to the screen, save it in a file, or pipe it down to another tool.

4.5.2 Cewl (Custom Word List generator)

Cewl[6] is a wordlist generator whose referenced words are crawled and scraped from a website. It is particularly useful for penetration testers or security professionals when they want to generate any word lists that will have specifications toward a target, like password cracking or even other social engineering purposes.

Cewl can crawl the target website for words on its pages and compile them into a wordlist. The depth of the crawl is configurable to specify how many levels of links it should follow. Cewl can also display the frequency of each word found so users know which words are most common on a target site.

Besides this, Cewl can extract metadata from the web pages in general for inclusion into the wordlist. Other various parameters, like minimum word length and user agent string, can also be defined by the user if needed to further fine-tune how the wordlist is generated. It can handle words in multiple languages according to the content received from a target site. Cewl works by crawling a given website to extract words from the HTML content of pages.

Here's a step-by-step process to use Cewl:

1. Specify the target URL: This is the source or starting point for where to begin crawling; users specify this by providing a website URL.
2. Set crawl depth and options: You can set how deep to crawl in addition to other options to be used, such as a minimum word length and whether to use metadata or not.
3. Run Cewl: Run the Cewl command with these parameters. It will crawl the site for you, extract the words, and make a wordlist.
4. Review and use of the wordlist: The above wordlist may be used as it is further processed for whatever purposes, such as password cracking.

4.5.3 CUPP (Common User Passwords Profiler)

CUPP[7] is a wordlist generator designed specially to build custom wordlists based on the target's personal information. In this case, it will come in very handy while performing social engineering attacks since it will generate password lists that have been tailored to the habits and preferences of a certain individual.

CUPP is a tool that can generate a personalized wordlist through personal information: it makes use of names, birthdays, anniversaries, hobbies, pet names, and other details. In interactive mode, CUPP will prompt for words connected with the target to be used to generate a more personalized wordlist. Also, it has predefined profiles and common password patterns, which enable the quick generation of wordlists without much input from the user. CUPP can combine and permute the given data, increasing all kinds of possibilities regarding the wordlist.

It is still possible, though, to configure input data and settings further in order to get a more fitted wordlist. In addition, CUPP can merge existing wordlists and introduce

[6] `https://github.com/digininja/CeWL`

[7] `https://github.com/Mebus/cupp`

variations such as leetspeak and many other modifications to increase the quality of a wordlist.

CUPP gathers information about a target and then uses that information to construct a wordlist specifically tailored for that target. That procedure comes in the following steps:

1. Gather information: Collect personal information about the target, such as their name, birthdate, pet names, favorite hobbies, and other relevant details.
2. Information input: This information is requested by CUPP when in interactive mode. Alternatively, one can load predefined profiles or existing data files.
3. Generate wordlist: Based on the provided information, the wordlist generator of CUPP creates a wordlist considering several combinations, permutations, and usual patterns of human-made passwords.
4. Review and use wordlist: One can use this generated wordlist directly for a number of purposes, such as cracking passwords.

4.5.4 Summary

Tool	**Description**
Crunch	This is a wordlist generator tool used in making custom password dictionaries by specifying character sets, lengths, and patterns.
Cewl (Custom Word List generator)	A tool that generates custom wordlists by crawling websites and collecting keywords, often used to create dictionaries tailored to specific targets.
CUPP (Common User Passwords Profiler)	Generates wordlists based on personal information, making it particularly effective for targeted password guessing attacks.

Table 4.4: Summary of Popular Password Cracking Tools

4.6 Defenses Mechanism

4.6.1 Strong Password Policies

Password policies are essential for the security of online systems. They ensure that rules are put in place, which will allow the user to choose strong and hard-to-guess passwords. Indeed, much of the recent research done on various facets of password policies has elicited very useful insights into how they should be designed for security and usability.

According to Sahin et al., [24], the will of the administrators of the websites is one of the main determinants of how effective password policies will be. Their decisions are guided by various aspects that address security concerns, compatibility with their systems, usability based on experience, and compliance with recommended guidelines. For example, quite often, system administrators choose a certain password length and complexity, which considers their system limitations while balancing security and user convenience.

A study by Shay et al. [25] has assessed various password-composition policies for their effect on password security and memorability as a function of both password length and character requirements. In particular, their results indicate that requiring longer passwords, say 12 or 16 characters, with fewer character-class requirements, can be more secure and more usable than the traditional policy emphasizing complexity over length. This will help users to construct passwords that are hard to guess but easier to remember.

Tan et al. [30] provided empirical evidence through large-scale analysis regarding password policies in the wild. Their results show that while a lot of websites require strong or stringent password compositions, actual implementation varies greatly, and these rules are far from the latest best practices. The misalignment underlines how key it is to have one unified message that administrators can easily implement for assurance of security while ensuring usability.

There are challenges in setting effective password policies. First, the technical and logistical issues involve the deployment of new policies, one of the common challenges administrators come across. Compatibility constraints, especially knowing that database character limitations and software limitations frequently fail to allow the kind of passwords that can be implemented. Second, the establishment of much safer policies is hindered by outdated mental models and reluctance to change. Modern password policy is hard to deploy and maintain at the best level without better software support and educational means for administrators [24].

Guo et al. [9] proposed a dynamic, personalized password policy based on individual personality traits. This strategy was designed to enhance both security and usability by tailoring the policies to users' preferences and cognitive competencies. They showed in their experiment that DPPP could generate stronger passwords against guessing attacks; at the same time, usability is at least comparable with the traditional policy.

4.6.2 Hashing and Salting

Hashing and salting are two important and widely used techniques to make password security powerful. These approaches guarantee protection of the original passwords from a number of different attacks, including guessing attacks, even in the case when password databases have been compromised.

Hashing is a cryptographic procedure in which plaintext passwords are transformed into a fixed-length string of characters, making it look like random data. This process is one way: it is computationally impossible to reverse the hash and discover the original password. Common hashing algorithms are MD5, SHA-1, and SHA-256.

The first major advantage of hashing is its one-way nature; it actually makes sure that the password will not be easily recovered. Still, hashing in itself is not nearly enough to secure a password from all kinds of attacks. For instance, attackers may use rainbow tables to compare hash values against their corresponding plain text passwords. This then brings in the role of salting.

Salting is the process of performing additional operations on the original data before that data is hashed. In this case, a unique, randomly generated string of characters is added to each password before it becomes hashed. This guarantees that even if two users share the same password, their hashes will be different due to the presence of unique salts, which effectively prevents using precomputed tables to crack the passwords.

Ebanesar and Suganthi [28] showed the practical implementation of salted hashing in their work using the SHA-256 algorithm. They used a strong random generation of salts for each password, and thereby, the hashed values that were salted correspondingly. They have increased security to a feasible extent since every user's password gets concatenated with a different salt, so no attacker can use either brute force or dictionary attack efficiently in their application.

In 2019, Sutriman and Sugiantoro [27] established that there are several schemes for combining passwords and salts that enhance hash algorithm security. They had implemented different schemes of combining passwords and salts before applying the hash function. Such complex combinations of passwords and salts help to increase the difficulty of cracking hashes and eventually increase the level of security.

The study examined three different schemes:

1. hash(rearrangement(password, salt))
2. hash(rearrangement(password, salt) + salt)
3. hash(hash(rearrangement(password, salt)))

Scheme 2, whereby the salt is included after permutation of the password and salt combination, is highly secure since it prolongs the cracking time to the optimum. This illustrates the point that it is not just the salt that is added but the very structured way in which the way it is added brings about enhanced security.

Karrar et al. [11] explored the use of swapping elements in an array as part of a salting technique to enhance password security. Their approach involved rearranging passwords and salt characters to create more complex inputs for the hash function. This method adds an additional layer of security by making the hash values even harder to predict or reverse-engineer.

Finally, hashing and salting are two inseparable techniques in password protection. Although password protection is guaranteed with storage by hashing, salting creates a unique password for storage. These well-enforced strategies greatly reduce the risk of password-guessing attacks, consequently improving security overall.

4.6.3 Rate Limiting and Account Lockout

Password rate limiting and account lockout are two of the major security features against password-guessing attacks. These techniques can make a difference in mitigating the risk of unauthorized access by controlling how many login attempts an attacker may attempt within any given timeframe.

Basically, the principle of rate limiting ensures that only a specified number of attempts is allowed for successful login within a certain period. This security measure is put in place to help decelerate attacks in the event that attackers automate their activities using automated tools to crack your passwords. Rate limiting helps prevent password-guessing attacks by reducing the speed of logins against the system.

Lu et al. (2018), for instance, conducted a study on authentication rate-limiting mechanisms for 182 websites in the Alexa Top 500 list. Their results showed that 72% of these websites (131 out of 182) would permit frequent, failed login attempts without appropriate mechanisms to rate-limit them. In fact, a study showed how poor rate-limiting implementations are common and pointed out how strong a defense needs to be implemented. The researchers suggested a black-box approach to modeling and validation of rate-limiting mechanisms, showing that many websites allowed attackers to test passwords at rates that could lead to account compromise. They suggested that websites create much stricter rate-limiting policies in order to enhance security.

In addition to rate limiting, account lockout mechanisms turn off a user account temporarily based on the number of failed login attempts made. In this manner, attackers cannot keep guessing but are forced to wait or contact support to recover an account.

Liu et al. [15] have assessed the vulnerability of account lockout attacks at the organizational level and suggested various countermeasures against this vulnerability. Their study, in the context of Microsoft services, revealed that 58%–77% of the studied organizations exposed authentication portals to lockout attacks. The countermeasures are, therefore, very important to know—keeping in mind their easy performance and minimal requirements.

In view of these vulnerabilities, Liu et al. [15] proposed some lockout bypass mechanisms for legitimate users. The evaluation results of those solutions show very high effectiveness, and the overhead of the network and systems was minimal. These mechanisms include:

- Strict authentication pools: These create separate pools for the authentication attempts based on user behavior and network characteristics, so legitimate access can preclude lockouts.
- Residential Network Tokens: Using TCP cookies, requests from known residential networks can be authenticated so that under all circumstances, including attack, unlimited access is provided to legitimate end users' accounts.

- User-Supplied Tokens: This would involve embedding non-public values within usernames in a bid to differentiate the legitimate request from malicious attempts.

Advanced techniques, on the other hand, for example, DALock by Blocki and Zhang [4], fortify the conventional mechanisms of rate limiting and account lockout. It maintains awareness of password distribution through a strike count and a hit count, which considers the popularity associated with the frequency of incorrect passwords guessed. Frequency oracles model the popularity of passwords using the hit count and the strike count to drive these lockout decisions.

Simulations demonstrated that DALock also offers a much better security/usability trade-off when compared to traditional K-strike mechanisms (account locked after K failed attempts). By distinguishing common typographical errors from suspicious behavior, DALock lowers the success rate of guessing attacks without unreasonably locking out legitimate users.

In conclusion, rate limiting and account lockout are important parts of password security. Properly designed rate limiting and account lockout mechanisms, in combination with adaptive policies and more advanced techniques, such as DALock, would improve system security without usability implications. Therefore, reaching a balance between user convenience and blocking unauthorized access to organizational systems would protect them against denial-of-service attacks and create a safe yet friendly environment for authentication.

4.6.4 Multi-Factor Authentication (MFA)

Multi-factor authentication (MFA) is important in security strengthening. In this regard, MFA makes any access considerably more secure by requiring users to provide several forms of identification before access is granted.

MFA involves the use of two or more independent credentials from different categories: knowledge-based authentication (e.g., password), possession-based authentication(e.g., smart card), and biometric-based authentication (e.g., fingerprint). It is by these factors that MFA gains its inherent layered defense through their combination, which makes it much harder for an attacker to gain access.

One of the strengths of MFA is that it reinforces the reduction of risks in security breaches. According to Papathanasiaki et al., [20], MFA indeed mitigates vulnerabilities associated with single-factor authentication (SFA). In most scenarios, passwords alone turn out weak and are easily compromised, but a second factor coupled with them improves security considerably.

For example, even if the attacker does manage to get a user's password, a second factor (whether it be a physical token or biometric verification) will still be required to enable them to log in. This multilayer approach creates a critical layer of protection for information and protection of the integrity of systems.

Where the security benefits of MFA are obvious, their implementation and usability matter equally. Singla and Verma [26] underline that the effectiveness of MFA depends on how frictionless it can be integrated into the workflow of the user. If it creates too much friction in the process, users may find ways to bypass it, and this situation will undermine its security benefits.

To address this, modern MFA solutions try to strike a balance between security and usability. For instance, biometric methods, either fingerprint or face recognition, are user-friendly but very secure. These can be used quickly and with much ease, and they do not require any extra information to be memorized or an extra device to be carried.

Even with its advantages, MFA is not free from challenges. Probably the most important among these is usability issues, mainly when extra steps needed for authentication are perceived as inconvenient. According to Papathanasaki et al., [20], user acceptance plays an important role in the success of MFA implementations. Being secure and convenient must, therefore, be mandatorily designed into a system.

Another problem is integrating MFA within existing systems and applications. Singla and Verma [26] describe the complexities in diffusing MFA across different platforms and guaranteeing compatibility with a number of technologies; this quite often requires expensive resources and high-level expertise that may not be available in every organization.

Modern developments in MFA are heading toward supporting both security and usability, such as adaptive MFA. It adjusts the level of authentication based on the context of a login. These factors include a user's location, device, and standard use patterns to determine whether further verification is necessary or not.

According to Singla and Verma [26], adaptive MFA would further smoothen the user experience while keeping the high-security levels intact. For instance, in such a system, if one attempts it from trusted devices at usual locations, then probably one factor will be required. On the other hand, in case of attempts from unknown devices or locations, multiple factors would be required.

Multi-factor authentication is one of the most promising ways to secure digital systems, imperfections and all. Backed by multiple verification modes, MFA builds a very strong defense against unauthorized access, thus raising the security bar. However, careful implementation with a balance between security and usability is still required. It will be an adaptive, user-friendly MFA in times ahead that will play an important role in shielding sensitive information in a more digital world.

4.6.5 Summary

Defense Mechanism	**Description**
Strong Password Policies	Ensures that a long and complex password is necessarily used to reduce the probability of successful password-guessing attacks. Encourage the use of passphrases, avoiding predictable patterns.
Hashing and Salting	use of cryptographic hashing and adds salt to each password individually before hashing, so rainbow table attacks would not work anymore
Rate Limiting and Account Lockout	The constraint on the number of login attempts within a time period evades online brute-force attacks. Principally, account lockout or CAPTCHA techniques are used after a certain number of failed attempts.
Multi-Factor Authentication (MFA)	It will require other forms of verification, like security tokens or biometric verification, in addition to the password. This would bring down the risks associated with unauthorized access in case your password gets compromised..

Table 4.5: Defensive Mechanisms Against Password Guessing

Chapter 5

Role of PII in Password Guessing

5.1 Introduction of PII

During password guessing attacks, there is one important factor that really separates targeted from trawling, and this is the use of data specific to a user, variously called either personal information or personally identifiable information (PII). Personal information forms the basis necessary for making the targeted guessing attack more accurate and efficient. PII simply refers to any information that identifies an individual, and this may include data such as names, birthdates, addresses, phone numbers, and even social media information.

Personal information can be divided into three major categories [32], each having a different level of secrecy and relevance to the passwords:

1. Personally Identifiable Information (PII): This category represents *semi-public* data such as a name, birth date, and phone number. These are typically known to friends, colleagues, and acquaintances but private beyond these circles. PII can further be broken down into:

   - Type-1 PII: These are the basic building blocks of passwords, such as names and birth dates, and users often set them in as parts of their passphrases.

   - Type-2 PII: like gender and education which might influence password creation behaviors, but which are not used in the password themselves

2. User identification credentials: These could be usernames and passwords. A username would be the end user's public identification, while a password would be considered private and secure.

3. Other personal data: Generally not used for the guessing of passwords, such as the person's employment records or financial data.

Attackers make use of PII to guess passwords more efficiently and effectively. Most people, in most cases, develop their passwords using their personal information. This implies that if this information is compromised, it is a simple guess of their password. For example, a person's name and date of birth can be used to derive passwords such as "John1985" or "JaneDoe123.".

Also, users tend to reuse or slightly modify their passwords across different sites. In fact, sister passwords may be easily retrievable by the attacker due to data breaches [10], making the benefits of targeted guessing attacks generally worse.

In other words, the role of PII in targeted password-guessing attacks is big. To that end, highly targeted guesses regarding personal details will then be made by the attackers, raising the possibility of success to a larger extent. In that regard, users and organizations should always be aware of these risks. We can minimize the threats that emanated previously from the misuse of PII in password guessing through strong password practices and being careful with what we share online.

5.1.1 Summary

Aspect	**Description**
PII	information that can directly or indirectly identify an individual, including names, birthdates, addresses, phone numbers, and social media information. Attackers normally use PII to target guessing attacks in order to increase the chances of success in password guessing.
Types of PII	There can be two types of PII: Type-1 and Type-2 PII. Type-1 holds the basic blocks of PII that are usually used in a password. Examples include names, birth dates, etc. Type 2 includes information like gender and education that affect password formation behavior but are not used in passwords as such.
PII and Password Reuse	With users usually either just repeating or only slightly modifying passwords at hundreds of sites, compromised PII will make password guessing much easier since the attackers can predict the variations from the available personal information.

Table 5.1: Overview of PII in Password Guessing

5.2 Password Guessing Frameworks based on PII

5.2.1 Personal-PCFG

Personal-PCFG was proposed in 2016 by Li et al. [14] and is based on the basic idea of the PCFG method developed by Weir et al. [34] (described in **section 2.5.2**) and provides an extension to further improve its efficiency.

Instead of only using L, S, and D symbols, it uses more semantics symbols like, for example: "B" for birthdate, "N" for name, "E" for email address, "A" for account name, "C" for cellphone number, and "I" for ID number.

Personal-PCFG contains 4 phases [14]:

- Personal Information Matching: For every password, the algorithm searches for its presence or substring in a user's personal information. The length of each matched segment is recorded, and each unmatched segment of the password is left as it is. For instance, if Alice's birth date is 16th August 1988 and she uses her password to log "helloalice816!" then in the matching phase, the word "Alice" will be replaced by "N_5" and "816" would be replaced by "B_3". The other part that is left is "hello." Hence, the resultant string will be "$helloN_5B_3$!".

- Password Pre-processing: This stage is similar to the original PCFG pre-processing routine but eliminates segments matched to personal information from additional analysis. The output of the personal information matching stage then gets incorporated back to update the password structure. For instance, the string "$helloN_5B_3$!" is then converted into "$L_5N_5B_3S_1$".

- Guess Generation: The procedure of this step is similar to the underlying PCFG, whereby all "D" and "S" symbols are replaced with some actual strings of words seen in the training set. "L" symbols are replaced with words from a dictionary. Guesses are continually formed without waiting for all possible guesses to be computed and sorted. For instance, the phrase "$L_5N_5B_3S_1$" gets converted to "$helloN_5B_3$!" if the very first 5-letter word in the dictionary is "hello" and the most probable special symbol is "!" having only one character. In this regard, it is assumed that all words with the same length would have an equal probability within the dictionary.

- Adaptive substitution: Unlike the original PCFG, Personal-PCFG requires guesses to be further instantiated with user-specific personal information. Each personal information symbol is replaced with the corresponding user-specific information of the same length. Multiple candidates of the same length are included for trial if available. For instance, in "$helloN_5B_3$!", the substring "N_5" is replaced with "Alice." For "B_3", multiple substrings of "19880816" are tried, resulting in guesses like "helloalice198!," "helloalice988!,"..., "helloalice816!" Each candidate guess is tested until one matches the target password.

Personal PCFG outperforms regular PCFG significantly, both in online and offline attacks. For instance, for the moderate-sized 500,000-guess corpus, Personal-PCFG demonstrates the same recognition rate that the original PCFG does with over 200 million guesses. It will be only efficient across various tests insofar as Personal-PCFG will be capable of cracking a good number of passwords with a restricted number of guesses. The numbers in the results point out that Personal-PCFG can crack up to 4.8% of passwords within just five guesses compared to only 0.9% for the original PCFG.

5.2.2 TarGuess-I

Targuess-I [32] is a model to guess a user's passwords online using type-I PII, for instance, name and birthdate. The approach of the authors is primarily based on the algorithm by Weir et al. [34] using PCFG. This has already been significantly successful in password-guessing trawling scenarios. In its training process, PCFG sees that these passwords are made up of letter, digit, or symbol segments. For instance, the password "iloveyou@1314" can be parsed to obtain the L-segment "iloveyou", the S-segment "@", and the D-segment "1314" to form the basic structure $L_7S_1D_4$. Similarly, "wanglei1982" is then parsed into $L_7S_1D_4$.

The TarGuess-I algorithm introduces type-based PII tags like N_1, B_1, or N_7. The subscript number defines the subtype of the PII tag. For instance, N_1 uses the full name, whereas an abbreviation of the full name uses N_2, and B_1 refers to a full birthdate in year-month-day format. This technique refines conventional PCFG by the inclusion of PII segments in a manner supporting more personalization in guessing. This gives rise to a PII-enriched context-free grammar $\mathcal{G_I} = (\mathcal{V}, \Sigma, \mathcal{S}, \mathcal{R})$,where [32]:

1. $\mathcal{S} \in \mathcal{V}$ is the start symbol.
2. $\mathcal{V} = \{\mathcal{S}; L, D, S; N_1, , N_7; B_1, , B_{10}; A_1, A_2, A_3; E_1, E_2, E_3; P_1, P_2; I_1, I_2, I_3\}$ is a finite set of variables, where:
    - N: Name
        - N_1: full name (e.g., johndoe)
        - N_2: abbreviation of the full name (e.g., jd)
        - N_3: last name (e.g., doe)
        - N_4: first name (e.g., john)
        - N_5: 1st letter of the first name + last name (e.g., jdoe)
        - N_6: for last name + the 1st letter of the first name (e.g., doej)
        - N_7: last name with its 1st letter capitalized (e.g., Doe)
    - B: Birthday
        - B_1: full birthday in YMD format (e.g., 20000101)
        - B_2: full birthday in MDY (e.g., 01012000)
        - B_3: full birthday in DMY (e.g., 01012000)
        - B_4: date in birthday (e.g., 0101)
        - B_5: year in birthday (e.g., 2000)
        - B_6: Year+Month (e.g., 200001)
        - B_7: Month+Year (e.g., 012000)
        - B_8: last two digits of year + date in MD format (e.g., 000101)
        - B_9: date in MD format + the last two digits of the year (e.g., 010100)
        - B_{10}: date in DM format + the last two digits of the year (e.g., 010100)

- A: Account name
  - A_1: full account name (e.g., jdoe123)
  - A_2: the (first) letter segment of account name (e.g., jdoe)
  - A_3: the (first) digital segment of account name (e.g., 123)
- E: Email
  - E_1: full email prefix (e.g., johndoe00)
  - E_2: first letter segment of email prefix (e.g., johndoe)
  - E_3: first digital-segment of account name (e.g., 00)
- P: Phone number
  - P_1: full number (e.g., 0123456789)
  - P_2: first three digits (e.g., 012)
  - P_3: last four digits (e.g., 6789)
- I: Chinese National Identification Number
  - I_1: last 4 digits
  - I_2: first 3 digits
  - I_3: first 6 digits

3. $\Sigma = \{95 \text{ printable ASCII codes, Null}\}$ is a finite set disjoint from $\mathcal{V}$ and contains all the terminals of $\mathcal{G}_{\mathcal{I}}$.
4. $\mathcal{R}$ is a finite set of rules of the form $A \to \alpha$, with $A \in \mathcal{V}$ and $\alpha \in \mathcal{V} \cup \Sigma$

Such a guessing process will exploit the longest prefix rule and incorporate all valid PII segments learned during the training phase to achieve possible accuracy. For example, assuming that john06071982 is matched to John Smith's account name "john0607", under the longest-prefix rule, it will parse A_1B_5 and not N_3B_2. Provided that abbreviations occur on the birth date, the corresponding tag will be replaced by the full segment so that both full and abbreviated data segments are created. For example, both "john06071982" and "john671982" will be generated provided that the pattern N_4B_2 is employed for password guessing.

These tags based on types of PII can be easily extended to support any type of PII. For example, if a researcher wants to consider new semantic usages like a hobby, then corresponding type-based tags like H for a hobby can be added. During the generation phase, the guessing algorithm will not parse passwords with tags and symbols undefined in the training data to make the model robust.

TarGuess-I is evaluated against the 12306 dataset. It can account for a target user given only the email; name and birthdate are very successful in TarGuess-I's guessing of a target user's password at an average chance of 20.18% within the first ten guesses. It drops to 13.61% with name and birthday only. This proves that email, account name, and birthdate are of high value regarding online attacks, tremendously increasing the success rate compared to Personal-PCFG.

By combining type-based PII tags and PCFG, TarGuess-I gives a more accurate and efficient model for password guessing. It outperforms the previous models across many scenarios.

5.2.3 Targuess-II

TarGuess-II [32] is a method for guessing a given user's password, PW_x, at a given service using his sister password, PW_s, at another service. This problem is quite challenging because of the usually small number of allowed guesses in online attacks and the large existing number of transformation rules applicable to passwords.

TarGuess-II is data-driven. Here, it reuses password pairs: a password leaked from some other similar service and the target password. The underlying concept is to match email and username pairs, creating valid non-identical sister password pairs. This creates a set of transformations applied to PW_s to generate possible PW_x guesses.

Now, the training process of TarGuess-II is to measure how likely PW_s is derived from PW_x. If PW_s happens to be a popular password, then this increases the likelihood of matching. The technique encompasses parsing passwords into segments like L, D, and S and further applying several kinds of transformations in the line of insertion, deletion, capitalization, and string movement.

1. Segment-level Training: This first step is done in calculating the segment similarity of PW_A and PW_B by implementing structure-level insert and delete rules using the Levenshtein distance (LD) algorithm to modify the segments.
2. Training Transformations: Training of several transformation rules can be done, such as:
    - Insertion/Deletion: Training includes rules at both the structure and segment level. (e.g., $L_3D_3 \rightarrow L_3D_3S_2$ or even L_3 : abc $\rightarrow L_2$: bc))
    - Capitalization: Training includes odds on various types of capitalization. (e.g., $abc123 \rightarrow Abc123$)
    - Leet: Training includes odds on various types of leet transformation. (e.g., L_1: a $\leftrightarrow$ @)
    - String Movement: Looks for the movement of substrings in the password. (e.g., $abc123 \rightarrow 123abc$)
    - Reverse Operations: Training to reverse segments or the entire password. (e.g., $abc123 \rightarrow 321cba$)

5.2.4 Targuess-III

TarGuess-III [32] guess a user's password by exploiting one of the user's sister passwords alongside some PII. Actually, this is realistic since, after knowing a user's sister's password, an attacker can obtain most of the PII, like email addresses or even names. This method focuses mainly on type-1 PII uses, such as name and birthday-related cues.

While more available information may sometimes complicate guessing due to the challenge of having few guesses available, it usually makes guessing more complicated. Suppose an attacker is to guess the eBay password of Alice Smith. The password has to be at least eight characters. An attacker knows that Alice was born in 1978 and that one of her passwords that was revealed after the leak from Yahoo was "Alice1978Yahoo.". They might make guesses like "Alice1978eBay," "Alice1978," and "12345678." Determination of which to mention first needs an adaptive model that includes PII.

Targuess-III achieves this by extending the grammar used in TarGuess-II with PII-based tags. This means it is a PII-enriched, password reuse-based grammar. It includes six types of PII usages as with Targuess-I and adds a number of type-based PII tags. In the training phase of TarGuess-III, the parsing of passwords annotated by PII tags undergoes execution before applying six structure-level transformation rules. During the guess generation phase, terminals and pre-terminals are produced by the grammar. Pre-terminals are intermediate guesses containing PII-based tags.

Suppose Alice's password at Yahoo was "Alice1978Yahoo." TarGuess-III is able to parse this into something such as $N_5B_5W_1$, where N_4 stands for the first name, B_5 for the birth year, and W_1 for the website. This structure could then create a guess like "Alice1978eBay" with high probability because it entails minimal transformations.

Experiments have shown that TarGuess-III performs very well. Even in the cases where a user does not reuse exactly the same passwords among different services, TarGuess-III can still achieve a success rate as high as 23.48% within 100 guesses. Besides, out of those passwords that TarGuess-III failed to guess, over 80% of them have a huge difference from the original one.

5.2.5 Targuess-IV

TarGuess-IV [32] is a rich password-guessing model that enhances its power by integrating both type-1 and type-2 PII with a sister password of the user. This method solves the problem of integration of type-2 PII, such as gender, which cannot be directly measured in the classical sense through methods such as PII tag-based PCFG grammars or Markov n-grams. In this regard, TarGuess-IV makes use of Bayesian theory and by proving the following theorem:

Let pw denote the event that the password pw is selected by U for a service, pw' denote the event that pw' was selected by U for another service and was leaked, and A_i (where $i = 1, 2, \ldots, n$) denote one kind of user PII attributes, including both type-1 and type-2 ones. The theorem states:

$$\Pr(pw|pw', A_1, A_2, \cdots, A_n) = \frac{\prod_{i=1}^{n} \Pr(pw|pw', A_i)}{\Pr(pw|pw')^{n-1}},$$

Under the assumptions that $A_1, A_2, \cdots, A_n$ are mutually independent and that they are also mutually independent under the events pw and (pw, pw').

The key insight with TarGuess-IV is that the blend of type-1 and type-2 PII increases the rate of password-guessing successes. Given a sister password to that user from another service, together with multiples of PII attributes, a model can make better guesses. This theorem applied in TarGuess-IV details how to compute the probability of guessing a user's password given the inclusion of those PII attributes and thus represents a "divide-and-conquer" approach for the integration of multiple types of PII.

It is also reflected in the improvements of TarGuess-IV over its previous versions. For instance, these are 24.51% and 30.66% with 100 and 1,000 guesses correspondingly when this type-2 PII, like gender, is used by TarGuess-IV to achieve very remarkable success rate improvements. This already outperforms TarGuess-III due to the consideration it has for type-2 PII.

This clearly devolves to the fact that including more personal information within the algorithm very highly improves the attack success rate in password guessing, making TarGuess-IV a very effective tool.

5.2.6 Targuess-I-H and TarMarkov-H

Targuess-I-H

In addition to type-based PII tags from Targuess-I [32], Targuess-I-H [6] use HFS as a new grammar tag. Its grammar $G_{TarGuess-I-H} = (S, \mathcal{V}, \Sigma, \mathcal{R})$ is described as below by Dong et al. [6]:

1. $S \in \mathcal{V}$ is the start symbol;
2. $\mathcal{V} = \{S; L_n, D_n, S_n; N_n, B_n, A_n, E_n, I_n, P_n; H_n^i; \epsilon\}$ is the set of grammar tags, where
   (a) L_n, D_n, S_n are the grammar tags of basic PCFG [34], representing the letter, digit, and symbol strings of length n, respectively;
   (b) $N_n, B_n, A_n, E_n, I_n, P_n$ are the grammar tags of TarGuess-I [32], representing the different forms of Name, Birthday, User name, Email, ID number, and Phone number distinguished by the number n;
   (c) H_n^i is proposed in the paper [6] for the *first time*, representing the set of strings ranked i among those substrings of length n in descending order of frequency;
   (d) ϵ is the terminator;
3. Σ is the set of 94 printable ASCII characters.
4. $\mathcal{R}$ is a finite set of rules of the form $A \to \alpha$, with $A \in \mathcal{V}$ and $\alpha \in \mathcal{V} \cup \Sigma$

TarMarkov-H

TarMarkov is a sequence model predicting the next state of a string conditioned on its current state at time t. Dong et al. [6] introduced a new state, HFS, into TarMarkov and a new targeted password probabilistic model entitled TarMarkov-H. Its grammar $G_{TarMarkov-H} = (S, \mathcal{V}, \Sigma, \mathcal{R})$ is described as below by Dong et al. [6]:

1. $S \in \mathcal{V}$ is the start symbol;
2. $\mathcal{V} = \{S; N_n, B_n, U_n, E_n, I_n, T_n, H_n^i, \Sigma; \epsilon\}$ is the state set, where
   (a) $N_n, B_n, U_n, E_n, I_n, T_n$ and H_n^i have the same meaning as the corresponding grammar tags in TarGuess-I-H, except that they represent different states in TarMarkov-H;
   (b) Σ is the set of 94 printable ASCII characters;
   (c) ϵ is the terminator;
3. $\mathcal{R}$ is a finite set of markov state transition rules of the form $s_1 \to s_2$, with $s_1, s_2 \in \mathcal{V}^*$.

HFS

In a password dataset, HFSs are password substrings with a frequency exceeding a certain threshold, and they can be identified by taking the following steps [6]:

1. Record the count $C(p_s)$ of each password substring p_s with the length $n \geq 3$;
2. Set the threshold T_1 and delete the substrings with a count less than T_1;

3. Modify the substring count record as

   $C(p_s)^{new} = C(p_s)^{old} - \sum_{c \in \Sigma} [C(c + p_s)^{old} + C(p_s + c)^{old}]$,

   where $C(p_s)^{old}$ is the original count record of p_s, and $c + p_s$ and $p_s + c$ respectively mean that the character c is concatenated to the beginning and end of p_s;
4. Set the threshold T_2 and identify p_s as a HFS if $C(p_s)^{new} \geq T_2$;
5. Store HFSs with the same length n into the set $\mathcal{H}_n (n \geq 3)$, arrange them in descending order of count, and denote the set of substrings ranked i in $\mathcal{H}_n$ as $\mathcal{H}_n^i$.

5.2.7 Summary

Framework	**Description**
Personal-PCFG	An extension of the method of probabilistic context-free grammars, this framework personalizes information such as names and birth dates as part of the password structure, hence increasing the efficiency in guessing passwords tremendously by using semantics specific to the individual.
TarGuess-I	A framework designed to utilize Type-1 PII for generating the targeted password guesses. It exploits common PII elements, like names and birth dates, to improve the probability of guessing.
TarGuess-II	Focused on guessing a user's password based upon knowledge of a sister password from another service. Different types of transformations, according to personal information, generate all possible passwords for this target service.
TarGuess-III	Makes targeted guesses by combining a user's sister password with PII. That is, this approach uses the structure of known passwords but generalizes them using PII-specific tags, thereby improving guessing accuracy.
TarGuess-IV	Represents an extension of past TarGuess frameworks to front-run more complex transformations and added PII data types for even more precise password guessing.
TarGuess-I-H	A further improvement on TarGuess-I by incorporating into the grammar an HFS tag that ranks password substrings based on their frequency of use, improving guess accuracy by focusing on commonly used substrings more.
TarMarkov-H	A model that puts a combination of Markov models with the High-Frequency Substring approach to build a sequence model, which is going to predict what the next likely password substring will be, given frequency and the current state, making highly probable password guesses.

Table 5.2: PII-Based Password Guessing Frameworks

Chapter 6

Project's mission, objectives and requirements

In this project, we create a password generation tool using PII and a detailed analysis tool for the PII inside passwords. We aim to improve accuracy and efficiency in guessing passwords by making use of large PII datasets and adapting to ever-changing password policies and user habits.

6.1 Requirements

6.1.1 List of requirements and how they relate one to another

1. PII datasets: These will form the basis of modeling and analysis and, hence, our password generation tool. The choice of the training dataset is crucial since the guessing tool only relies on this. So, it is important to choose a training dataset that uses the same policy as the targeted website policy.
2. Data processing capabilities: The processing of these sets needs treatment and analysis of a high order; this is the essence of merging PII into our algorithms.
3. Password policies and user behavior: We want to understand better how these interact in reality to adapt our tools for real-world use cases, thus impacting their design and functionality.
4. Integration frameworks: These would, of course, be necessary to integrate PII into password-guessing algorithms.
5. Testing environments: These testing environments are needed to test the efficiency and accuracy of the developed tools for their practical applicability.
6. An analysis tool that produces statistics and graphs on the usage of PII in passwords
7. A password generation tool based on an advanced guessing model.

6.1.2 Requirements covered by state of the art

1. There is a large amount of leaked PII and password datasets on the internet
2. Password policy and human behavior have been the subject of much research these last years (**section 2.4.3** and **section 2.4.4**)
3. PII password generation tool to test against: This is already covered through tools like CUPP (**section 2.5.4**).
4. Advanced guessing models: The TarGuess-I-H framework make good model for this (**section 2.6.2**).

5. I will use my personal computer as a testing environment

6.1.3 Requirements not covered by state of the art

1. Handling huge and diverse PII data sets: Resulting from this is the lack of efficient handling provided by existing tools when handling huge and diverse sets of PII datasets.
2. There is a lack of tools for PII analysis within password security.
3. There is no wordlist generation tool based on PII using advanced techniques

6.2 Project scoping

6.2.1 Mission statement of this project

The mission of this project is to develop and create one PII analysis tool and a PII-based password generation tool to use as a wordlist in a password-guessing tool like Hashcat. We want to improve the precision and performance of already existing PII password generation tools by several folds. We will use enormous PII datasets to get the best probabilities for guessing passwords.

6.2.2 Explicit out-of-scope definition

- Non-PII-Based Password Guessing Techniques: The thesis will only be concerned with PII-based approaches and password structure probabilities.
- Other than type-1 PII: We only identify type-1 PII since we only implement the basis of a more developed tool.

Chapter 7

Implementation

7.1 Methodology

7.1.1 Implementing Basis for Tools

High Frequency Substring (HFS) Generation

To effectively use the Targuess-I-H method for password guessing, we first need to implement the HFS generation, as detailed in **section 2.6.2**. This step is important because HFS will help rank the most likely password candidates based on their structure and frequency in a given dataset. This is implemented in the file hfs.py.

Generating PII Variations

Once the HFS generation is in place, the next step involves generating all possible variations of Personally Identifiable Information (PII) as follows:

- N: Name
  - N_1: full name
  - N_2: abbreviation of full name
  - N_3: last name
  - N_4: first name
  - N_5: 1st letter of the first name + last name
  - N_6: for last name + the 1st letter of the first name
  - N_7: family name with its 1st letter capitalized
- B: Birthday
  - B_1: full birthday in YMD format
  - B_2: full birthday in MDY
  - B_3: full birthday in DMY
  - B_4: date in birthday
  - B_5: year in birthday
  - B_6: Year+Month
  - B_7: Month+Year
  - B_8: last two digits of year + date in MD format
  - B_9: date in MD format + the last two digits of year

  - B_{10}: date in DM format + the last two digits of year

- A: Account name
  - A_1: full account name
  - A_2: the (first) letter segment of the account name
  - A_3: the (first) digital-segment of account name
- E: Email
  - E_1: full email prefix
  - E_2: first letter segment of email prefix
  - E_3: first digital-segment of account name
- Ph: Phone number
  - Ph_1: full number
  - Ph_2: first three digits
  - Ph_3: last four digits

In addition to these PII variations, we add H_n^i representing the set of strings ranked i among those substrings of length n in descending order of frequency that we get from the previously implemented HFS generation. This part is implemented in file structure.py.

Identifying Password Segments

Now, with the generation of HFS and PII variations, we can easily identify PII and non-PII (L, S, D, P, H) segments within a password. This segmentation will allow us to map the structure of a password, which forms the basis of the password-guessing tool that we seek to implement. After doing that, we already have almost everything for our PII analysis tool. This is also implemented in file structure.py.

Training the Password Generation Tool

Now, having the basis for our guessing methodology, we move toward implementing the training part of our password generation tool. To begin with this task, a training dataset is further needed. This dataset will be used to generate three dictionaries (letters, specials, and digits) and a list of different password structures, along with associated probabilities. That is the frequency at which each structure occurs in the dataset with respect to the total size of the dataset.

How to Generate New Passwords

Now that we have the dictionaries and structure-probability pairs, the generation of new passwords based on a set of PII is possible. This will incessively be the taking of each structure-probability pair to create a list of new pairs by replacing each segment of the structure systematically with corresponding elements from our PII variations and HFS dictionary along with other relevant dictionaries.

Statistics Computation

Now that we have everything but statistics computation to build our analysis tool, we have to define what statistics we want to compute and show to users.

First, we'll compute some statistics about the length of the passwords in the dataset:

1. The length distribution in the dataset
2. The minimum length in the dataset
3. The maximum length in the dataset
4. The average length in the dataset
5. The standard deviation

Then, we compute statistics on the different types of characters in the password dataset:

1. The average distribution of letters in a password in percentage
2. The average distribution of digits in a password in percentage
3. The average distribution of specials in a password in percentage
4. The most common characters in passwords
5. The percentage of passwords containing letters
6. The percentage of passwords containing digits
7. The percentage of passwords containing specials
8. The percentage of passwords containing uppercase letters
9. The percentage of passwords containing lowercase letters

After that, we implement every function to compute statistics on the distribution of PII types and sub-types in passwords:

1. Distribution of passwords (not) containing at least one PII subtype.
2. Distribution of PII types in passwords containing PII
3. Distribution of PII subtypes in passwords containing the corresponding PII type

Finally, we compute statistics on password structures in PII-based and non-PII-based passwords:

1. Distribution of structures
2. Average structure length (number of segments)
3. Structure diversity (the number of unique structures in the dataset)

7.1.2 Main Tools

Now that we have everything we need, we can start implementing the main tools with the previously implemented functions.

Analysis Tool

To run the tool, the user opens the terminal in the project directory and runs one of the following commands:

- run by taking dataset in a mysql database: `python analyzer.py mysql --host <host> --user <username> --password <password> --database <database> --table <table> --fields <fields_list>`.

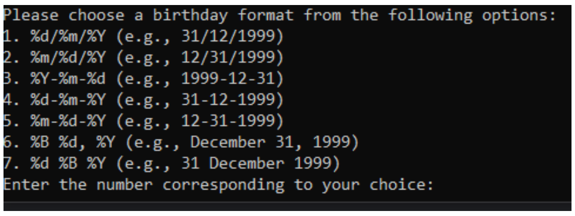
  With `<host>` the MySQL host, `<username>` the MySQL username, `<password>` the MySQL password, `<database>` the database name, `<table>` the table name in the database, `<fields_list>` the list of field to extract from the database (the valid fields are: "firstname", "lastname", "username", "birthday", "email", "phone", "password") in addition to these options the user can use optional options:

  1. `--condition <condition>` with `<condition>` some conditions for SQL query (e.g., "'username'='John'")
  2. `--additional <additional>` with `<additional>` some additional SQL clauses (e.g., "LIMIT 1000")

- run by taking dataset from a file: `python analyzer.py <path_to_dataset> --delimiter <delimiter>`

In addition to these options, the user can choose the file to use as an HFS dictionary with the option `--hfs <path_to_hfs>` with `<path_to_hfs>` the path to the HFS dictionary

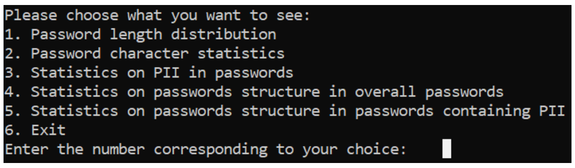
If the user runs the tool with a file dataset, then he has to choose the birthday format used in the given file (**Figure 7.1**).



```
Please choose a birthday format from the following options:
1. %d/%m/%Y (e.g., 31/12/1999)
2. %m/%d/%Y (e.g., 12/31/1999)
3. %Y-%m-%d (e.g., 1999-12-31)
4. %d-%m-%Y (e.g., 31-12-1999)
5. %m-%d-%Y (e.g., 12-31-1999)
6. %B %d, %Y (e.g., December 31, 1999)
7. %d %B %Y (e.g., 31 December 1999)
Enter the number corresponding to your choice:
```

Figure 7.1: Birthday format choice

After the birthday format has been chosen, the user sees the main menu, where he can put the number associated with the statistics he wants to explore, or he can exit the tool. (**Figure 7.2**)



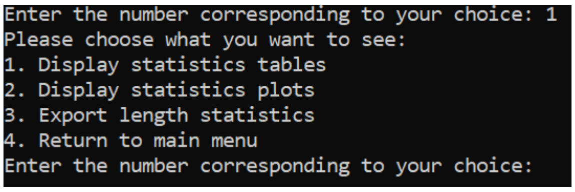
```
Please choose what you want to see:
1. Password length distribution
2. Password character statistics
3. Statistics on PII in passwords
4. Statistics on passwords structure in overall passwords
5. Statistics on passwords structure in passwords containing PII
6. Exit
Enter the number corresponding to your choice:
```

Figure 7.2: Main menu

Once the user makes his choice, he has four options (**Figure 7.3**):

1. Display statistics tables (see tables content in next section)
2. Display statistics plots (see different plots in next section)

3. Export the statistics in CSV format
4. Return to the main menu



```
Enter the number corresponding to your choice: 1
Please choose what you want to see:
1. Display statistics tables
2. Display statistics plots
3. Export length statistics
4. Return to main menu
Enter the number corresponding to your choice:
```

Figure 7.3: Statistics menu example - Password length distribution menu

When the user finishes what he wants to do, he can go back to the main menu and exit the tool.

Password Generation Tool

To run the tool, the user opens the terminal in the project directory and runs one of the following commands:

- run by taking training dataset in a MySQL database: `python generator.py mysql --host <host> --user <username> --password <password> --database <database> --table <table> --fields <fields_list>`.

  With `<host>` the MySQL host, `<username>` the MySQL username, `<password>` the MySQL password, `<database>` the database name, `<table>` the table name in the database, `<fields_list>` the list of field to extract from the database (the valid fields are: "firstname", "lastname", "username", "birthday", "email", "phone", "password")

- run by taking dataset from a file: `python generator.py <path_to_dataset> --delimiter <delimiter>`. The used file should contain the fields name in the first line, separated by the delimiter (the valid fields are: "firstname", "lastname", "username", "birthday", "email", "phone", "password").

In addition to these options, the user can:

- Choose the probability threshold for the created wordlist, i.e., we take every guess with a probability of success greater than the threshold. The user uses the option `--threshold <threshold>` with `<threshold>` the probability threshold (between 0 and 1). The default threshold is 0.00001.

- Choose the path for the output file with the option `--output <output>` with `<output>` the path to the output file. The default output is in the output directory, with the filename being the first name of the target.

- Add a condition to the SQL query with the option `--condition <condition>` with `<condition>` some conditions for SQL query (e.g., "'username'='John'")

- Add additional restriction to the SQL query with the option `--additional <additional>` with `<additional>` some additional SQL clauses (e.g., "LIMIT 1000")

- Choose the file to use as HFS dictionary with the option `--hfs <path_to_hfs>` with `<path_to_hfs>`, the path to the top 100 HFS dictionary. The top_hfs_500_50.json file in the default/hfs directory is used by default

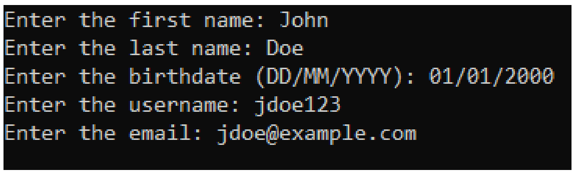
- Choose the special characters dictionary with the option `--s_dictionary <path_to_s_dictionary>` with `<path_to_s_dictionary`, the path to the special character dictionary. The default used file s_dictionary.json is in the default/dictionary directory

- Choose the digits characters dictionary with the option `--d_dictionary <path_to_d_dictionary>` with `<path_to_d_dictionary`, the path to the digits characters dictionary. The default used file d_dictionary.json is in the default/dictionary directory

- Choose the letters characters dictionary with the option `--l_dictionary <path_to_l_dictionary>` with `<path_to_l_dictionary`, the path to the letters characters dictionary. The default used file l_dictionary.json is in the default/dictionary directory

As for the analysis tool, if the user runs the tool with a file dataset, he has to choose the birthday format used in the file (**Figure 7.1**).

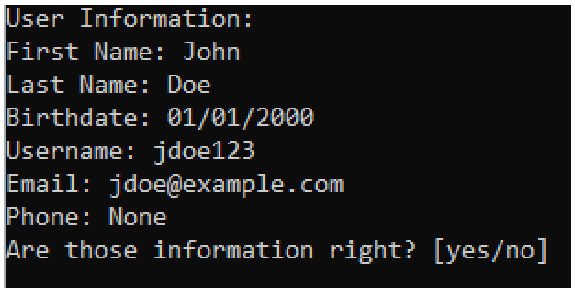
After that, the user is asked to enter the target personal information according to the ones used in the training dataset (e.g., if the training dataset does not contain the phone field, then the user is not asked to enter the target phone number **Figure 7.4**)



```
Enter the first name: John
Enter the last name: Doe
Enter the birthdate (DD/MM/YYYY): 01/01/2000
Enter the username: jdoe123
Enter the email: jdoe@example.com
```

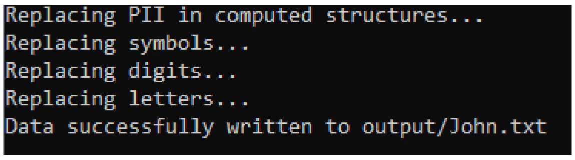
Figure 7.4: Target personal information

Once the user enters every asked personal information, the tool asks him for validation (**Figure 7.5**) before proceeding to the dictionary generation (**Figure 7.6**).



```
User Information:
First Name: John
Last Name: Doe
Birthdate: 01/01/2000
Username: jdoe123
Email: jdoe@example.com
Phone: None
Are those information right? [yes/no]
```

Figure 7.5: Target personal information validation



```
Replacing PII in computed structures...
Replacing symbols...
Replacing digits...
Replacing letters...
Data successfully written to output/John.txt
```

Figure 7.6: Dictionary generation

At the end of the generation, a new file is created with the generation content.

7.1.3 Secondary Tools

Besides the main tools that I developed, I also implemented some secondary tools that can be very useful in supporting the main tools.

HFS Generation Tool

The aim of the HFS Generation Tool (hfs_dictionaries.py) is to take a dataset and return the top 100 substrings from that dataset. These substrings categorize frequently occurring strings of characters users more often include in their passwords.

To run the tool, the user opens the terminal in the project directory and runs the following command:

- `python hfs_dictionary.py <path_to_dataset> <output_file>`. With `<path_to_dataset>` the path to the password dataset and `<output_file>` the path to the file to create. The used file should contain one password per line.

In addition to this, the user can:

- Choose the thresholds as described in **section 2.6.2** with options `--T1 <first_threshold>` and `--T2 <second_threshold>`. By default, they are set to 500 and 50, respectively.
- Choose the minimum and maximum length for the substrings with options `--min <minimum_length>` and `--max <maximum_length>`. By default, they are set to 3 and 8, respectively.

After running the tool, a file is created to the specified output path with the top 100 hfs in the used dataset.

Dictionaries Generation Tool

The other supporting tool that I developed is the Dictionaries Generation Tool (dictionaries.py). This tool is used to generate dictionaries from a training dataset, which are in turn fed to the wordlist generation tool as an essential input for it to work. The dictionaries generated by this tool will be used to map correctly the components that compose the passwords throughout the phase of guessing.

To run the tool, the user opens the terminal in the project directory and runs one of the following commands:

- run by taking training dataset in a mysql database: `python dictionaries.py mysql <output_s_dictionary_path> <output_d_dictionary_path> <output_l_dictionary_path> --host <host> --user <username> --password <password> --database <database> --table <table> --fields <fields_list>`.

  With `<output_s_dictionary_path>` the output path of the special characters dictionary, `<output_d_dictionary_path>` the output path of the digits characters dictionary, `<output_l_dictionary_path>` the output path of the letters characters dictionary, `<host>` the MySQL host, `<username>` the MySQL username, `<password>` the MySQL password, `<database>` the database name, `<table>` the table name in the database, `<fields_list>` the list of field to extract from the database (the valid fields are: "firstname", "lastname", "username", "birthday", "email", "phone", "password")

- run by taking dataset from a file: `python analyzer.py <path_to_dataset> <output_s_dictionary_p` `<output_d_dictionary_path> <output_l_dictionary_path> --delimiter <delimiter>`. The used file should contain the fields name in the first line, separated by the delimiter (the valid fields are: "firstname", "lastname", "username", "birthday", "email", "phone", "password").

In addition to these options, the user can:

- Choose the file to use as HFS dictionary with the option `--hfs <path_to_hfs>` with `<path_to_hfs>`, the path to the top 100 HFS dictionary. The top_hfs_500_50.json file in the default/hfs directory is used by default

The tool generates three kinds of dictionaries, all sorted by length and then by frequency:

1. Specials Dictionary: This is the dictionary keeping all sequences of special characters associated with their frequency in the training dataset.
2. Digits Dictionary: This dictionary consists of all numeric characters associated with their frequency present in the dataset.
3. Letters/Words Dictionary: The dictionary consists of all single or sequence of letters and frequent words from the training dataset.

These dictionaries are created and written into the respective files. Later on, they can be imported into the generation tool.

Chapter 8

Experimentation preparation & data collection

8.1 Setup, Requirements, Environment, Tools & Materials

8.1.1 Hardware and software setup

For the experiment, I used my personal computer, which has the following system:

- CPU: 12th Gen Intel(R) Core(TM) i5-12400F, 2.50 GHz
- GPU: NVIDIA GeForce GTX 1050 Ti
- RAM: 16,0 Gb
- Storage: 930 Gb
- OS: Windows 10 Pro 22H2
- Python 3.12.

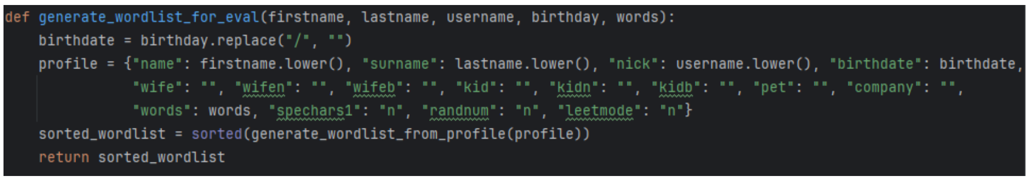
8.1.2 Tools and libraries

- Pycharm for development
- GitLab as version control
- Python libraries. These libraries can be installed with the command `pip install -r .\src\requirements.txt` in the project repository.
    1. argparse
    2. datetime
    3. mysql.connector
    4. tqdm
    5. statistics
    6. matplotlib
    7. tabulate

8.2 CUPP Tool

First, we need to install the CUPP repository from github[1].

[1]`https://github.com/Mebus/cupp`



```python
def generate_wordlist_for_eval(firstname, lastname, username, birthday, words):
    birthdate = birthday.replace("/", "")
    profile = {"name": firstname.lower(), "surname": lastname.lower(), "nick": username.lower(), "birthdate": birthdate,
               "wife": "", "wifen": "", "wifeb": "", "kid": "", "kidn": "", "kidb": "", "pet": "", "company": "",
               "words": words, "spechars1": "n", "randnum": "n", "leetmode": "n"}
    sorted_wordlist = sorted(generate_wordlist_from_profile(profile))
    return sorted_wordlist
```

Figure 8.1: Added function in cupp.py

To use the wordlist generation methodology used by the CUPP tool, we change the function `generate_wordlist_from_profile` to return `unique_list_finished` and we write a function in cupp.py named `generate_wordlist_for_eval` to get the generated dictionary in a list format instead of a .txt file so we can use it in our evaluation.

8.3 Data Source

For my experiments, I used different datasets:

1. A leaked database from 2016 as a training and test dataset. This dataset contains user information such as first name, last name, username, birth date, email, and plain text passwords.
2. The rockyou.txt[2] file containing passwords to build the HFS dictionary

Since the database file is a .sql file, I had to set up a mysql database. For that, we have to install mysql server[3]. After installation, run mysql in the terminal with the command `mysql -u <USERNAME> -p`, with <USERNAME> the mysql username.

Then, create an empty database with the SQL command `CREATE DATABASE <DB_NAME>;`, with <DB_NAME> as the name of the database.

When this is done, exit mysql and enter the following command in the terminal: `mysql -u <USERNAME> -p <DB_NAME> < <PATH_TO_DATABASE_FILE>` with <USERNAME> the mysql username, <DB_NAME> the name of the database, and <PATH_TO_DATABASE_FILE> the path to the sql databasee file.

8.3.1 Leaked database data preprocessing

First, since my tool only accepts a defined set of field names, I had to change some column names of the user table before importing it into my database. The column names are changed as follows:

- first_name → firstname
- last_name → lastname
- date_of_birth → birthday

After the database is in place, we extract the firstname, lastname, username, email, birthday, password of every row in the user's table that satisfies the following restrictions:

1. Each field must be different from "NULL"

[2]https://www.kaggle.com/datasets/wjburns/common-password-list-rockyoutxt
[3]https://dev.mysql.com/downloads/installer/

2. Each field must not be empty
3. The birth date must be $\geq$ '1900-01-01' and $\leq$ to the current date

This gives us 2,210,953 rows of the table. After gathering these rows from the table, we still apply another restriction: each field must be decodable in 'utf-8' (excepted birth date, which has type datetime.date)

We now have the complete dataset, which contains 2,210,941 rows, for our experiment.

Chapter 9

Experiment's output/Data Analysis

9.1 Database dataset analysis

Before doing experiments with the dictionary generation tool, we'll first analyze the database dataset with our analysis tool.

9.1.1 Length distribution

The distribution of password lengths detailed in **tables 9.1, 9.2** and **figure 9.1, 9.2** reveals that the most common lengths range between 6 and 10 characters, with the peak frequency at 8 characters (524,627 instances). Notably, passwords shorter than 11 characters dominate the dataset. The average password length is 8.73 characters, with a standard deviation of 2.35, indicating a moderate variance in length. The minimum recorded password length is 6 characters, while the maximum length is 25 characters, even though passwords with 21 characters and more are rare.

Table 9.1: Length distribution

Length	**Count**
6	405774
7	284504
8	524627
9	313603
10	285214
11	148242
12	99317
13	53425
14	36354
15	22623
16	15056
17	7466
18	5282
19	3303
20	6136
21	1
22	2
23	1
25	11

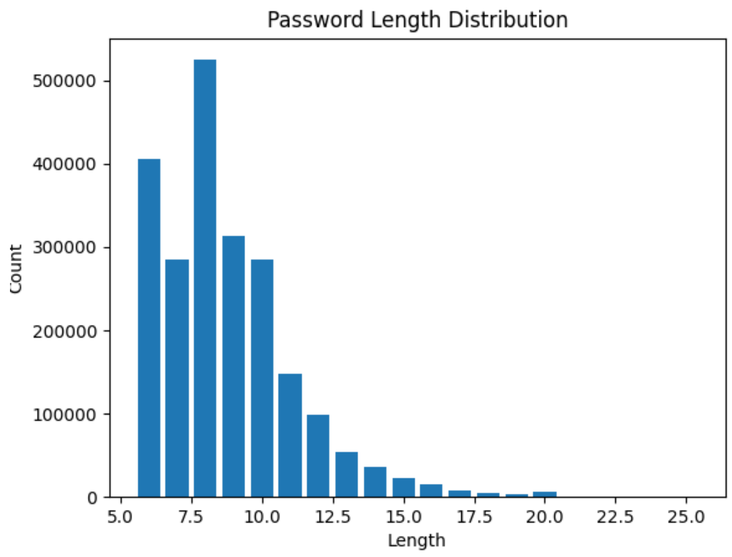


Figure 9.1: Passwords length distribution in database dataset

Table 9.2: Length statistics

Statistic	**Value**
Minimum Length	6
Maximum Length	25
Average Length	8.73
Standard Deviation	2.35

9.1.2 Character distribution

The **table 9.3** and **figures 9.3, 9.4, 9.5** gives us information about character statistics.

The analysis of password characters reveals some trends in how characters are used. The most flagrant and obvious observation is that the majority of passwords (82.09%) contain letters, with each password containing an average of 65.52% letters. Among these, lowercase letters dominate, appearing in 80.25% of passwords, while uppercase letters are much less common, present in only 7.39% of passwords.

Numbers are also recurrent in passwords, found in 62.40% of passwords and contributing to an average of 33.90% of the character composition. Special characters, however, are rarely used. Indeed, a password contains an average of 0.56% of special characters and appears in just 3.94% of all passwords.

The most common characters across the dataset are 'a', '1', 'e', 'i', and 'n', with 'a' being the most frequent, appearing 1,653,590 times in the dataset. This suggests that while passwords can include a mix of characters, users prefer simpler, letter-dominated combinations.

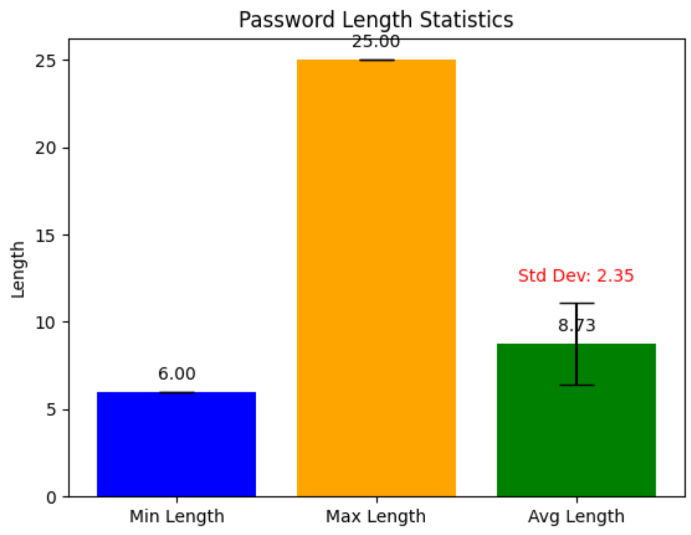


Figure 9.2: Passwords length statistics in database dataset

Table 9.3: Password character statistics

Statistic	Value
Average Percent Letters	65.52%
Average Percent Numbers	33.90%
Average Percent Specials	0.56%
Most Common Characters	a (1653590), 1 (1133517), e (1042851), i (907775), n (839879)
Percent Passwords with Letter	82.09%
Percent Passwords with Number	62.40%
Percent Passwords with Special	3.94%
Percent Passwords with Uppercase	7.39%
Percent Passwords with Lowercase	80.25%

9.1.3 PII distribution

In **table 9.4** and **figure 9.6**, we can see that there are 28.33% of passwords that contain PII, while the remaining 71.67% do not. So, approximately a third of the password database contains at least one PII information.

Table 9.4: Table for percentage of passwords containing PII in a set of passwords

Total	Count	Percentage
Containing PII	626282	28.33%
Not containing PII	1584659	71.67%

This breakdown highlights the patterns and prevalence of specific keys and sub-types within passwords that contain PII, providing important insights for improving password security.

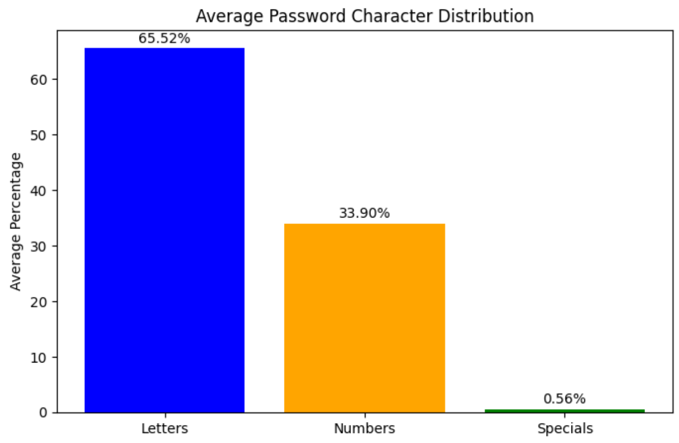


Figure 9.3: Average password character distribution in database dataset

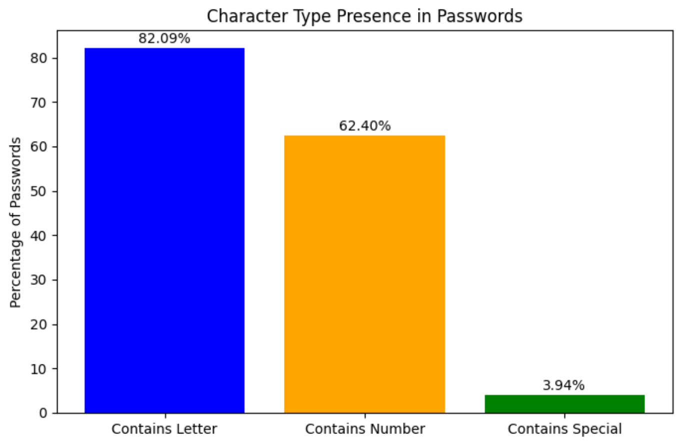


Figure 9.4: Character types presence in passwords from database dataset

From **table 9.5** and **figure 9.7**, "N" which is first name and last name variations, is found in 33.89% of the passwords containing PII, making it one of the most frequent. The key "E" for email variations is present in 9.88% of these passwords, indicating a lower but still significant occurrence. The key "A" for account name variations appears in 29.21% of the passwords, and "B", for birthdate variations, has the highest occurrence, being present in 35.90% of the passwords containing PII.

Table 9.5: Table for main keys presence in passwords containing PII

Main Key	**Total Count**	**Total Percentage**
N	212223	33.89%
E	61895	9.88%
A	182935	29.21%
B	224813	35.90%

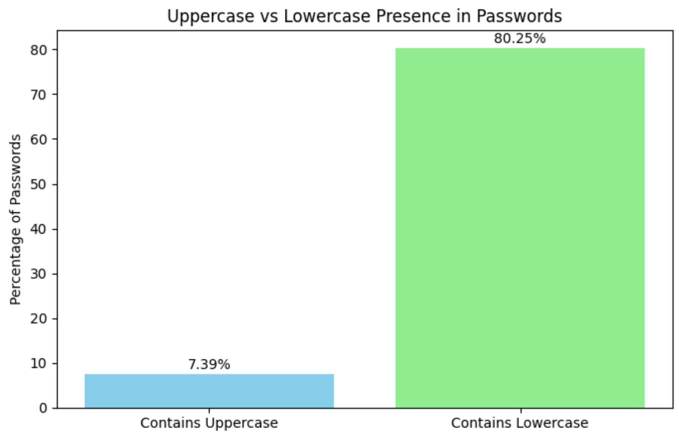


Figure 9.5: Uppercase vs. lowercase presence in passwords from database dataset

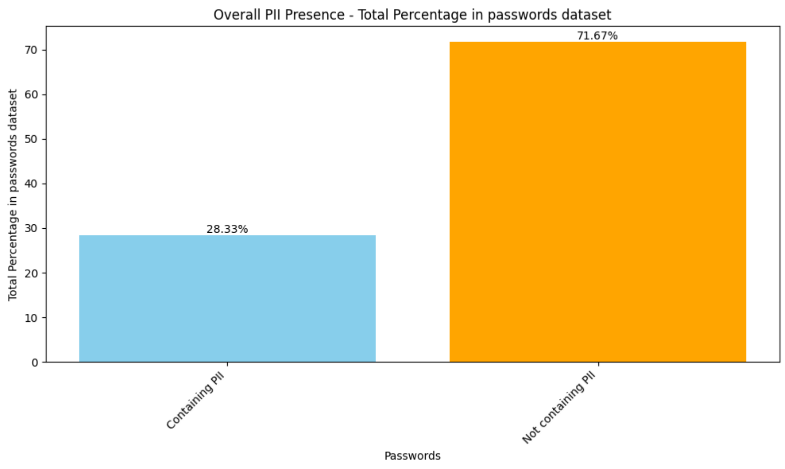


Figure 9.6: Overall PII presence in passwords from database dataset

For sub-types within these key, the **tables 9.6,9.7,9.8,9.9** and **figures 9.8,9.9,9.10,9.11** show that:

- In "N" sub-types "N4" (for first name) appears 43.13%, followed by "N3" (last name) (26.77%) and "N2" (full name abbreviation) (21.73%).
- In "A" sub-type "A3" (the (first) digital segment of account name) dominates with 59.23%, followed by "A1" (the (first) digital segment of account name) (30.09%) and "A2" (the (first) letter-segment of account name) (12.09%).
- In the "B" category, "B5" (year in birthday) is the most common sub-type. It appears in 58.83% of cases, with "B4" (date in birthday) (10.65%) and "B10" (full birthday in DMY) (8.44%) also notable.
- For "E," the sub-type "E3" (first digital segment of account name) is overwhelmingly the most common, accounting for 77.10%, while "E2" (first letter segment of email prefix) and "E1"(full email prefix) are less prevalent, at 12.65% and 11.00% respectively.

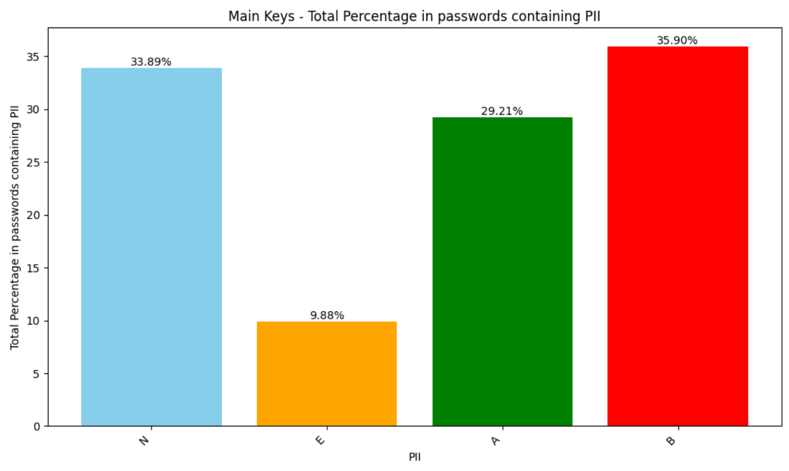


Figure 9.7: PII types in passwords containing PII from database dataset

Table 9.6: Table for A sub-types presence in passwords containing A

Sub PII	**Count**	**Percentage**
A1	55036	30.09%
A2	22113	12.09%
A3	108354	59.23%

9.1.4 Structures distribution

The most common structures in the general dataset of passwords (**table 9.10**) are L6, occurring 151,012 times and accounting for 6.83% of all occurrences; this is closely followed by L8, occurring 134,142 times and accounting for 6.07% of all occurrences; next in line is L7, with 112,216 occurrences and accounting for 5.07% of all occurrences; followed by L9, with 82,506 occurrences and accounting for 3.74% of all occurrences; lastly, D6 occurs 66,416 times, accounting for 3.00% of all occurrences. Other structures like L10, D10, letter-digit combinations like L6 D2, and so on, are also found but are less frequent. In **table 9.11** we can see that the average structure length is 1.86 on average for all passwords, while the total structure diversity comes to 97,803 unique structures.

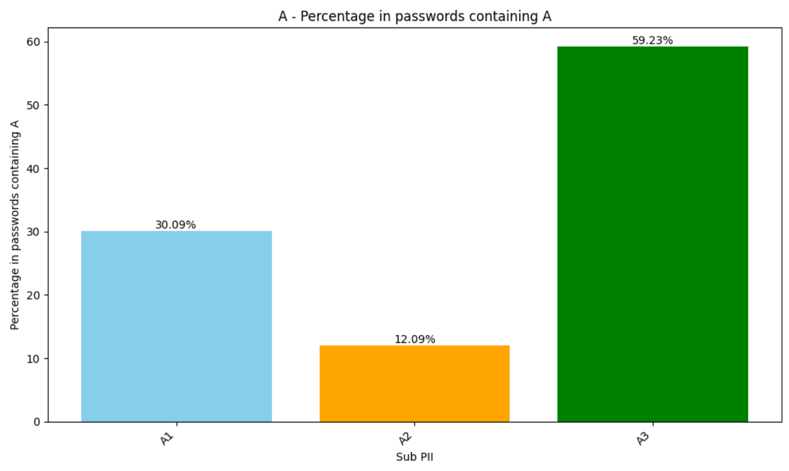


Figure 9.8: A sub-types distribution in passwords containing type A PII in database dataset

Table 9.7: Table for B sub-types presence in passwords containing B

Sub PII	Count	Percentage
B1	2949	1.31%
B10	18966	8.44%
B2	4860	2.16%
B3	17646	7.85%
B4	23947	10.65%
B5	132259	58.83%
B6	555	0.25%
B7	2394	1.06%
B8	8878	3.95%
B9	14556	6.47%

The most common structure among passwords with PII (**table 9.12**) is A1, which appears 34,941 times (5.58%). Other popular structures include N4 (2.42%), B3 (2.21%) and N3 (2.02%). Structures like L6 A3 and L7 A3 occur rather frequently, too, even though they are combinations of some sort. In **table 9.13**, we can see that the average structure length (number of segments in a structure) for passwords that contain PII is 2.28, and there are 55,056 different structures that contain PII.

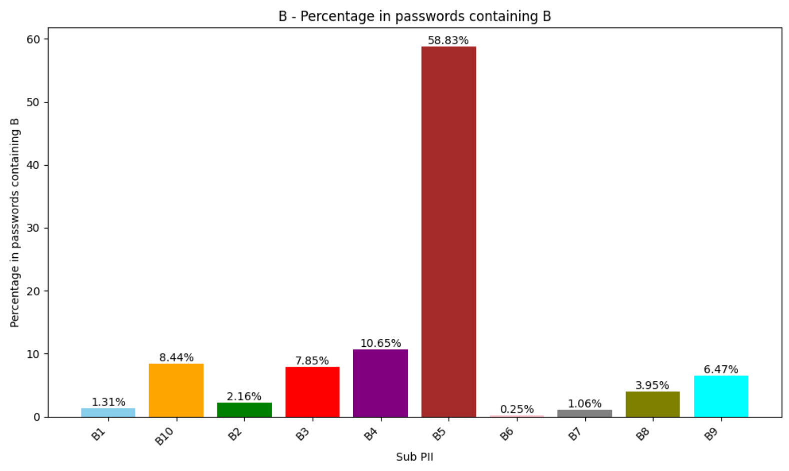


Figure 9.9: B sub-types distribution in passwords containing type B PII in database dataset

Table 9.8: Table for E sub-types presence in passwords containing E

Sub PII	Count	Percentage
E1	6810	11.00%
E2	7827	12.65%
E3	47724	77.10%

9.1.5 Analysis conclusion

First, the analysis gives us some critical insights concerning the database dataset. Most important is the fact that a large majority of passwords are short. In most cases, they are between 6 to 10 characters long, with 8 as the most common. This trend shows that users tend to create passwords of only the minimum required lengths and do not further use longer ones in order to enhance their security. Significantly, very few passwords have a length greater than 11 characters, suggesting ease of use as against the improved security gained with longer passwords.

The distribution of characters in the dataset clearly shows that letters are the most preferred in passwords, with special dominance by lowercase letters over uppercase letters. This pattern most likely suggests that, while users do include letters in their passwords, they tend not to use enough other available characters that would really help to increase the complexity of the passwords, such as uppercase letters and special characters. Special characters are rarely used and turn up in a very small percentage of all passwords, hence likely weakening the overall strength of passwords and increasing vulnerability to attacks.

Another important finding is that about 28.33 % of the passwords examined contain some form of personally identifiable information, which exposes those kinds of passwords to a high level of security risk since they are easy to guess or crack. The rest, 71.67 %, do not contain any PII, which would seem like good news. However, it is a serious concern that such a large number of these passwords contain PII. Most common PII elements include first and last names, birthdates, and account names, so it follows that most users would include easily guessable pieces of personal information when choosing passwords.

Finally, the structure of the passwords contributes, with the dataset having the majority of passwords made up of simple structures, such as only L6, L8, and L7, indicating basic

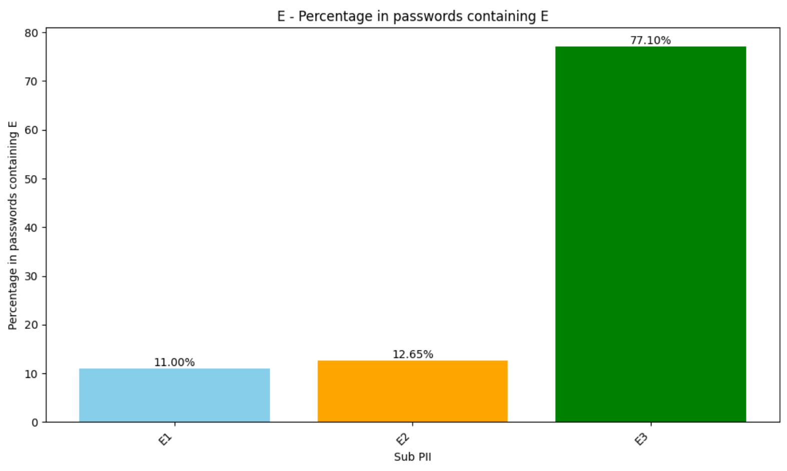


Figure 9.10: E sub-types distribution in passwords containing type E PII in database dataset

Table 9.9: Table for N sub-types presence in passwords containing N

Sub PII	**Count**	**Percentage**
N1	14247	6.71%
N2	46125	21.73%
N3	56807	26.77%
N4	91526	43.13%
N5	3193	1.50%
N6	2830	1.33%
N7	394	0.19%

letter sequences with predictable patterns. Passwords containing only digits and simple letter and digit combinations are also quite common. Despite the prevalence of these simple structures, this dataset is very diverse, with nearly 100,000 unique structures identified. That said, the mean structure length is relatively small, indicating that even within such diversity, many passwords aren't really complex.

Further password profiling with respect to PII-based passwords includes overly common structures, for example, those beginning with an account name segment or including first and last name segments.

These findings mean that a large number of users tend to create passwords that are far less secure than they could be, tending strongly toward too-short lengths, simple structures, and the inclusion of easily guessable personal information.

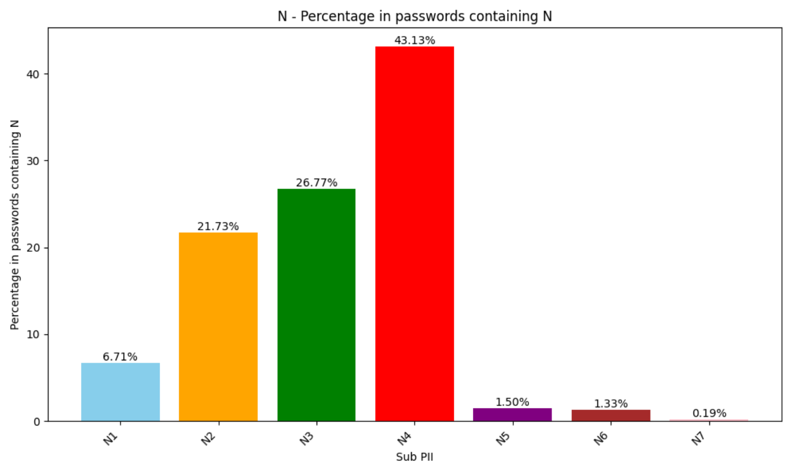


Figure 9.11: N sub-types distribution in passwords containing type N PII in database dataset

Table 9.10: Structure distribution in overall passwords

Structure	**Count**	**Percentage**
L6	151012	6.83%
L8	134142	6.07%
L7	112216	5.07%
L9	82506	3.74%
D6	66416	3.00%
L10	55604	2.51%
D10	44212	1.99%
L6 D2	38726	1.75%
D8	38218	1.73%
A1	34941	1.58%

9.2 Password Generation Tool Experimentation

9.2.1 Comparisons rules

To test our generation tool against the CUPP tool, we must define a fair way to compare the two tools. To do so, we follow the following rules:

1. We use the same training and testing dataset for the two tools
2. We use the same dictionaries (letters, digits, specials)
3. Since the CUPP tool is not designed to handle large datasets as input, we limit the dictionaries to 50 elements by length (e.g., a letters dictionary contains 50 elements of length six, and this is the same for every length)
4. We use the HFS dictionary in each tool
5. From the dataset analysis, we know that password length varies between 6 and 25, so we only generate passwords in this length range.
6. We also do not use the digits and symbols at the end options and leet speak option because it would generate too large dictionaries. So, we only compare our tool to the basic functionalities of CUPP.

Table 9.11: Structure statistics in overall passwords

Average Structure Length	1.86
Structure Diversity	97803

Table 9.12: Top 10 password structures in passwords containing PII

Structure	**Count**	**Percentage**
A1	34941	5.58%
N4	15183	2.42%
B3	13844	2.21%
N3	12656	2.02%
B10	11712	1.87%
L6 A3	11354	1.81%
N1	10389	1.66%
L6 B5	10362	1.65%
B9	9004	1.44%
L7 A3	7088	1.13%

Then, to run our comparison, we will use the database. However, since using the entire database would take too much time (more than 2 million rows to process for each iteration), we run ten iterations, each time with a randomly chosen training dataset of 5000 rows and a testing dataset of 1000 rows. This will still take some hours, but less than the entire database.

We randomly selected the training and testing dataset using ten different seeds: 245783, 918274, 562914, 83721

For the tool to generate dictionaries in moderate time, we use a probability threshold 0.000005, which means that we only take passwords with a higher probability than 0.000005

9.2.2 Comparisons metrics

We will compare the two tools on different metrics:

1. The average number of password guesses generated by the tools
2. The average success rate in guessing a password
3. The average number of passwords generated before a guess (rank)
4. The median of the number of passwords generated before a guess
5. The standard deviation of passwords generated before a guess
6. The number of successes below 1000 guesses
7. The time taken

9.2.3 Comparisons results

After running the evaluation that opposes the CUPP tool to the implemented generation tool, we get one table per iteration, a summary table, and some graphics in the "Gathered comparisons tables and figures" section.

From these tables, we can make the following observations:

Table 9.13: Structure statistics in passwords containing PII

Average Structure Length	2.28
Structure Diversity	55056

1. Average passwords generated per second: The CUPP tool consistently generates more than 1000 passwords per second with an average of 1017, while my tool consistently generates approximately 250 passwords per second with an average of 254.26.
2. Success rate: My tool consistently manages to guess more passwords than the CUPP tool successfully. Indeed, my tool managed to guess an average of 10.09% of the tested passwords, while the CUPP tool had an average success of 6.27
3. Average and median rank: The average rank (where a lower rank is better) and median rank for my tool are significantly lower than those for cupp. This means that the successful password guesses from my tool rank way better than those from the CUPP tool. More than that, we can see that my tool successfully guesses a password with a median of 92.95 guesses, while the CUPP tool manages it with a median of 763904 guesses (way, way more).
4. Standard deviation of rank: as showed by the median rank, the average standard deviation of both tools show that my tool has more consistency than the CUPP tool.
5. Number of successes below 1000 guesses: This shows that the CUPP tool never managed to guess a password with less than 1000 guesses, while most passwords my tool guessed are below 1000.

9.2.4 Gathered comparisons tables and figures

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	997565	8.7	635347	705679
evaluate_my_tool	81595.2	11.6	2492.27	98.5

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	318533	0	967.142
evaluate_my_tool	9299.11	90	328.213

(b) Rank Variability and Successes

Table 9.14: Iteration 1: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	1.01434e+06	5.7	694756	792364
evaluate_my_tool	108822	11.1	3046.5	52

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	313100	0	978.748
evaluate_my_tool	11328.4	80	375.469

(b) Rank Variability and Successes

Table 9.15: Iteration 2: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	987824	6.3	663018	772647
evaluate_my_tool	77974.2	9.3	4627.78	51

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	330328	0	988.943
evaluate_my_tool	12733	65	345.661

(b) Rank Variability and Successes

Table 9.16: Iteration 3: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	1.00824e+06	5.1	611482	755576
evaluate_my_tool	104111	9.4	2310.5	76

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	364392	0	989.586
evaluate_my_tool	4605.56	66	447.037

(b) Rank Variability and Successes

Table 9.17: Iteration 4: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	1.00412e+06	5.8	712772	836656
evaluate_my_tool	106258	10.5	4476.17	109

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	324942	0	1003.5
evaluate_my_tool	13925.5	71	468.799

(b) Rank Variability and Successes

Table 9.18: Iteration 5: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	987686	7.6	726477	846570
evaluate_my_tool	103569	10.1	3500.7	89

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	308719	0	971.4
evaluate_my_tool	9924.54	71	420.375

(b) Rank Variability and Successes

Table 9.19: Iteration 6: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	997924	5.9	546275	698602
evaluate_my_tool	93436.4	10.3	3023.94	119

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	363734	0	969.869
evaluate_my_tool	7632.61	66	399.001

(b) Rank Variability and Successes

Table 9.20: Iteration 7: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	992145	4.8	589915	695294
evaluate_my_tool	118360	8.4	2277.24	99

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	331356	0	983.584
evaluate_my_tool	4787.77	63	387.717

(b) Rank Variability and Successes

Table 9.21: Iteration 8: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	999785	5.7	613771	779590
evaluate_my_tool	89886.5	9.2	2025.24	144

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	354062	0	987.637
evaluate_my_tool	4976.53	65	363.315

(b) Rank Variability and Successes

Table 9.22: Iteration 9: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	990167	7.1	626376	756658
evaluate_my_tool	110297	11	3913.18	92

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	337626	0	974.032
evaluate_my_tool	15980.9	80	375.901

(b) Rank Variability and Successes

Table 9.23: Iteration 10: Comparison of Password Evaluation Methods

Method	Average passwords generated	Success Rate (%)	Average Rank	Median Rank
evaluate_cupp	997979	6.27	641929	763904
evaluate_my_tool	99430.9	10.09	3169.35	92.95

(a) Password Generation and Ranking

Method	Standard deviation of Rank	Number of Successes within Top 1000	Time (seconds
evaluate_cupp	334679	0	981.436
evaluate_my_tool	9519.4	71.7	391.149

(b) Rank Variability and Successes

Table 9.24: Average Results After Multiple Iterations

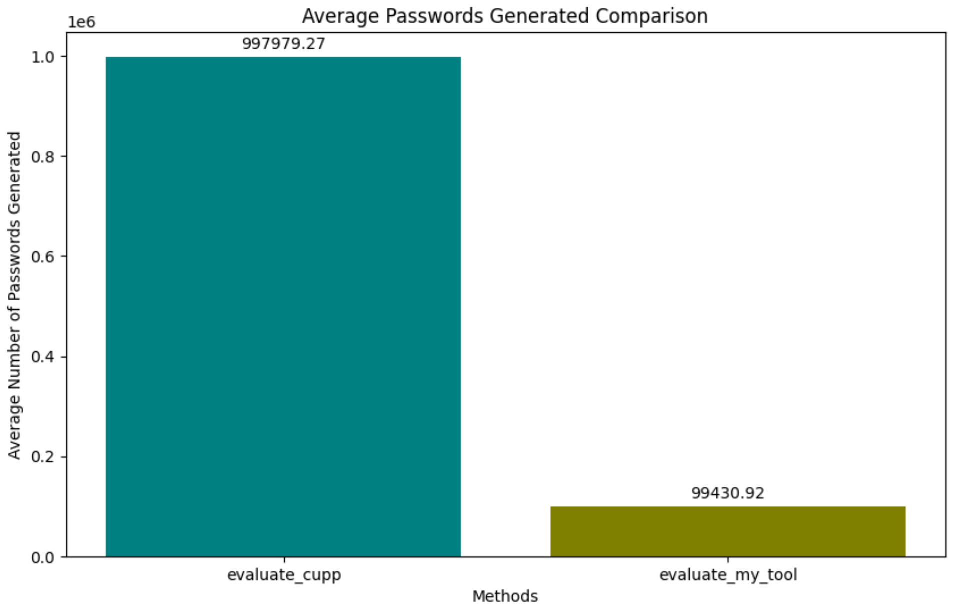


Figure 9.12: Average password generated per dictionary

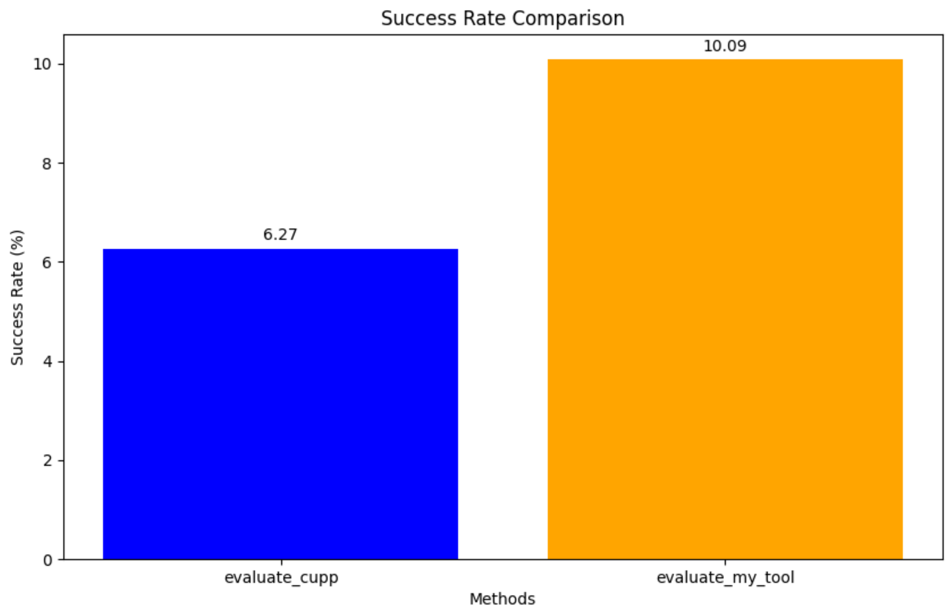


Figure 9.13: Average success rate

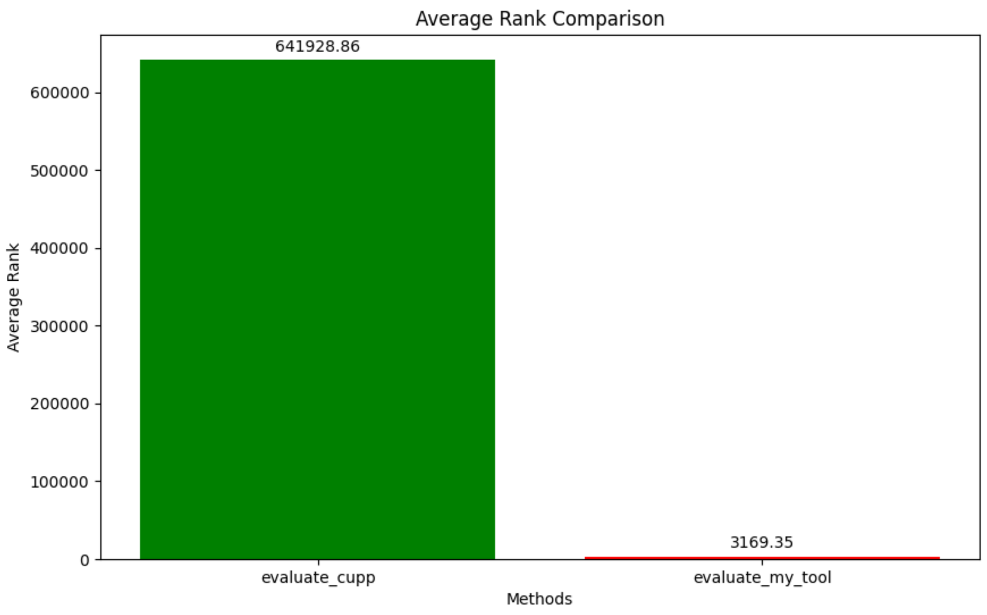


Figure 9.14: Average rank

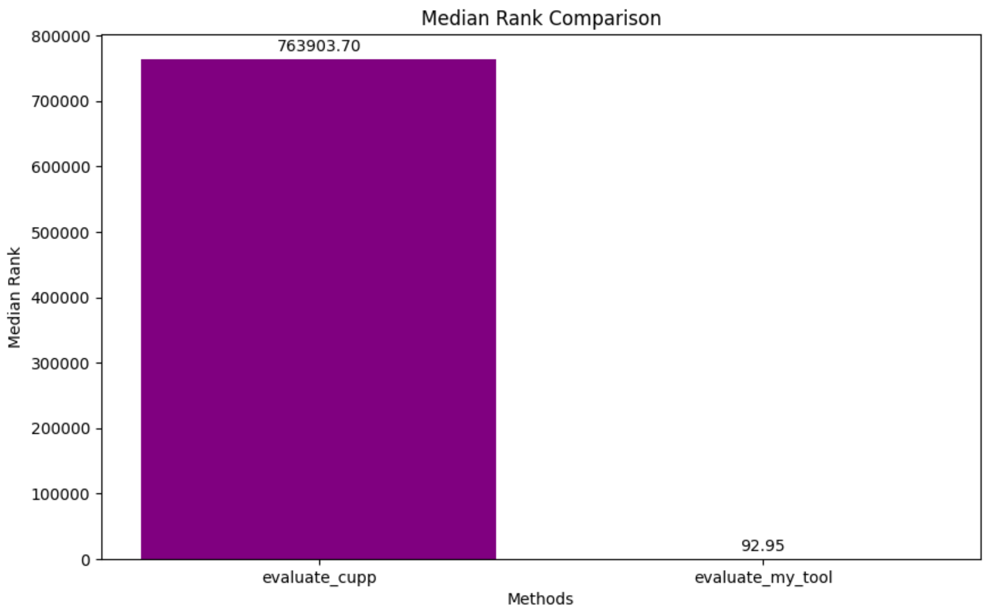


Figure 9.15: Average Median rank

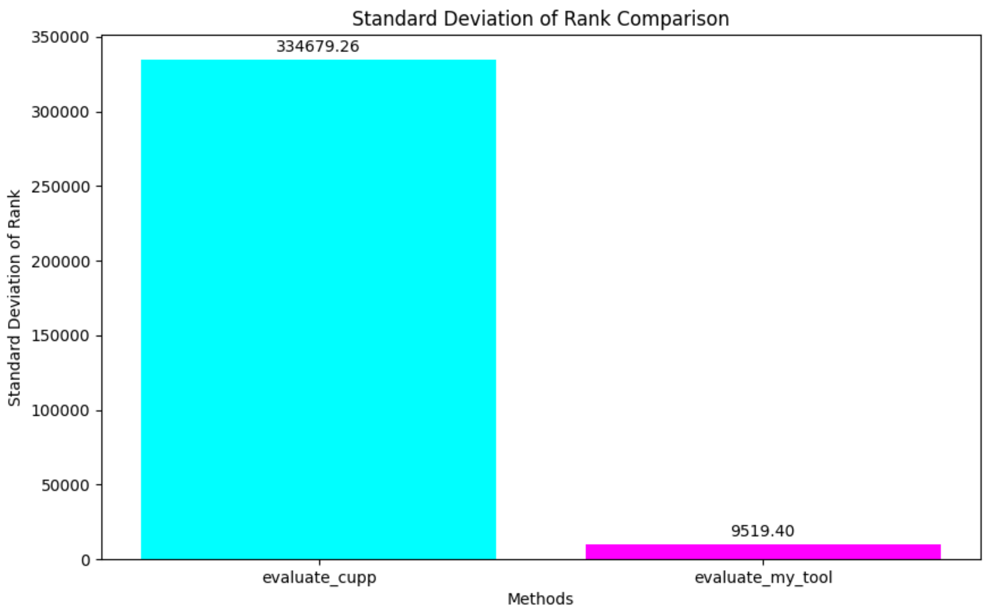


Figure 9.16: Average Standard deviation rank

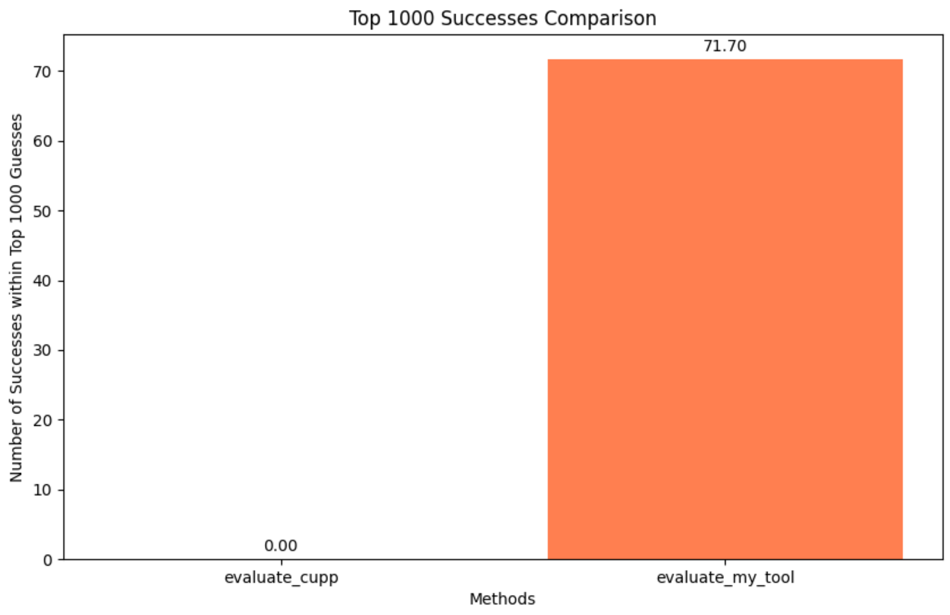


Figure 9.17: Average Number success

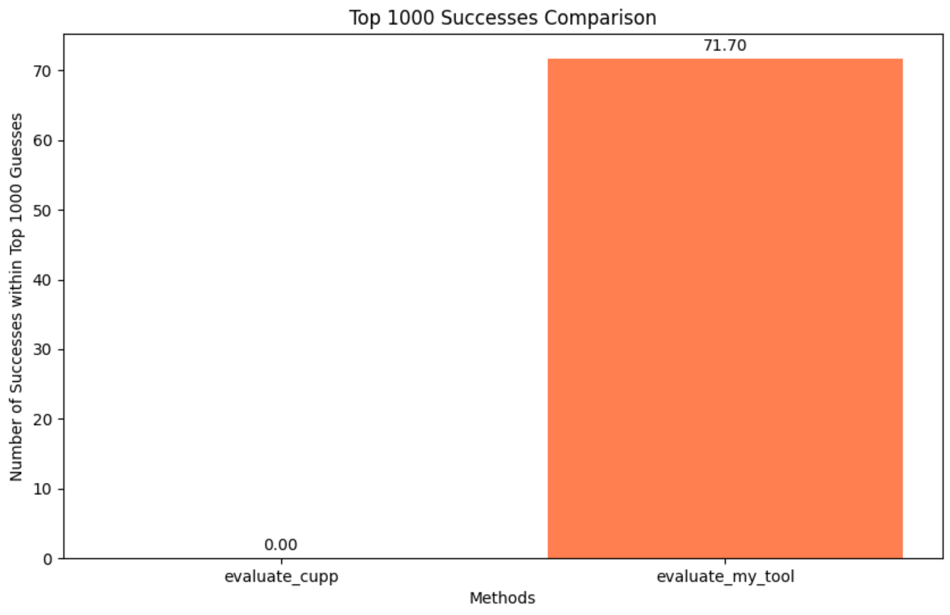


Figure 9.18: Average success below 1000 guesses

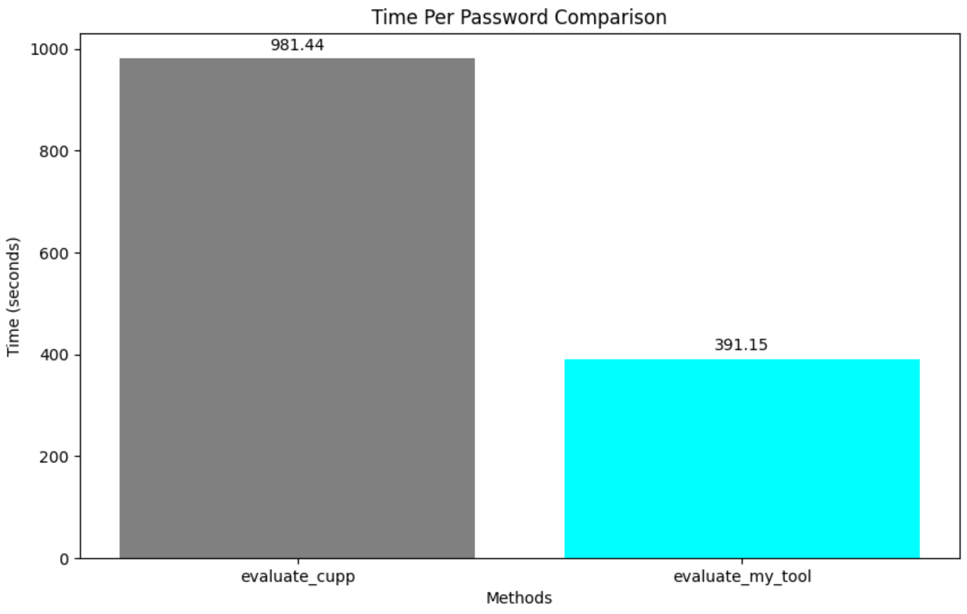


Figure 9.19: Average time taken to generate 1000 dictionaries

Chapter 10

Discussion

10.1 Comparison with state of the art/related works

This thesis is aimed at developing and evaluating a new password generation tool against one of the currently available tools, the CUPP (Common User Passwords Profiler) tool. Basically, this would aim to increase the efficiency and effectiveness of the methods of password guessing, especially in targeted attacks where personal information is important.

One of the most visible differences between the developed tool and CUPP is, therefore, the efficiency of the password generation part. For instance, while CUPP generates over 1 million passwords at each iteration, our tool makes the generation process considerably more refined with many fewer passwords: from 80,000 to as high as 118,000 in each iteration (with a probability threshold of 0.000005). Our tool, even though generating less quantity of passwords, terminated with a lot higher success percent that averaged at 10.09% in comparison to the cupp tool 6.27%. This illustrates that the refined approach reduces not only the number of attempts required but also increases the chances of one successfully cracking a password.

This can be attributed to the fact that our generated passwords are very near the target set. Our tool also performed better than CUPP for the average and median ranks. This great gain in rank effectiveness means it is better at ordering more likely words to the top of the list and, hence, more efficient in practical password guessing.

This was evidenced more by our tool's performance stability, with a lower standard deviation in the rank of guessed passwords compared to CUPP, which had a very high standard deviation of over 300,000, ranging between 4,600 and 15,900. Consistency is a very important property in cybersecurity applications since predictability and reliability of performance are the major considerations for these applications.

Our tool shows massive improvement over CUPP by integrating PII into the guessing and creating passwords. While CUPP broadly finds ways for password creation, our tool uses personal information to narrow guessing. With an era concentrating on technology, this comes in handy because most patterns of attackers deliver their ways clearly as a result of personal data at their disposal due to breaches or social engineering.

Results from this thesis can contribute to the prolonged view of security discussion on the use of brute force versus more focused attacks in cybersecurity applications. Proving that a more focused approach could bring better results with fewer guesses will further support the development of tools that rely on personal and contextual information to improve the hacking process. The improvements in success rates and efficiency that this thesis shows indicate that this avenue should always be pursued by future tools, even more so in sensitive data security. Thus, these features show that our tool improved relative to state-of-the-art solutions like CUPP in efficiency, effectiveness, and, more importantly, consistency.

This indicates new potential in adopting more refined and targeted approaches to the idea themselves in generating passwords, potentially radically improving cybersecurity.

10.2 Limitations of validity

10.2.1 Use of Basic Functionality of CUPP

The CUPP tool has multiple features, so one of the main limitations of this work was that only the basic ones were employed. The excluded features were options of digits and symbols and leet speak options to ensure a comparison on level ground with the developed tool; it may be presumed that this restriction affected the evaluation of overall performance. Where these could be relevant features in the real world, this comparison may not accurately reflect the relative strengths of the CUPP tool.

10.2.2 Limitation on Dictionary Size

Since the CUPP tool can not handle large datasets, we were doomed to limit the size of dictionaries to 50 elements per length. While this ensures that the tool will keep clear of performance problems, it simultaneously limits the generation process for passwords; an extended dictionary might give different results.

10.2.3 Probabilistic threshold and dataset size

The methodology adopted for this research was first and foremost influenced by time constraints. The practical need to constrain the size of the generated dictionaries meant that only a probabilistic threshold was included. It was secondly decided to train and test only 5,000 and 1,000 rows, respectively. While these choices have enabled the research to be completed in a given timeframe, they've also constrained its scope. Given more time, larger training and test datasets would have been possible, which would have made the results more robust and allowed greater insight into this tool's performance.

10.2.4 Dependency on a Single Leaked Dataset

The training and test datasets come from the same leak (that of the database, dated 2016). The dataset was picked since it holds enough variations of PII types that could be relevant for the analysis and generation tool. This is, however, a limit to the generalizability that can be derived from a single data set. The performance might be different if applied to more recent and diverse datasets. It should be validated against additional databases in order to validate the results.

10.2.5 Conclusion

These limitations thus somewhat attenuate the estimation of the generalizability of results. This tool was tested under conditions with limited dataset diversity and dictionary sizes. Given that real-world applications are characterized by highly prevalent password characteristics and user behavior diversity, effectiveness may differ. Furthermore, learning from only one data source could also give less than full weight to a tool's potential when applied over time in different environments and datasets. In sum, although this thesis has very useful contributions about generating or guessing passwords, one should keep these limitations at the back of one's mind. Recognizing these constraints is therefore useful not only to do justice to this research but also to open up new avenues for improvement in the future with respect to dataset diversity, dictionary size, and feature completeness.

Chapter 11

Future work

Most of the limitations identified in this thesis are primary focuses that will be addressed in future work. This includes the refinement of the tool concerning dependencies on certain datasets, small dictionary size, or more sophisticated handling of PII, as well as developing a regime for resolving these limitations to enhance the accuracy and reliability of the tool and its overall utility.

Since new forms of PII will always be emerging, it goes without saying that this tool should be updated to include these new types in the future. Further research involves integrating such PII variations into this tool to enhance the capacity to create effective and complete password dictionaries. Guessing strategies can be made very effective by noticing how personal information is used differently across cultures, languages, and regions.

The tool should perform better while generating the dictionaries. One may further implement several algorithms to improve the processing speed and reduce the resources being used by the system. This parallel processing could do quite well when making good use of the advanced hardware options available, such as GPU acceleration. This would greatly improve the efficiency of the tool and, hence, make it suitable for managing larger data sets.

This has to be relevant and up-to-date since the threats in cybersecurity are never static. An architecture, therefore, has to be provided in which datasets, algorithms, and the tool's user interface could be updated, potentially periodically, to handle new developments in password-guessing techniques or any other PII uses.

Future work needs to make this tool more effective and efficient. For example, it may entail adapting machine learning models via the evolving patterns of passwords or state-of-the-art heuristics to predict password structures more accurately with minimal input. This could be very helpful, provided there are studies about user evaluations of its practical effectiveness in real-world settings.

The tool could be taken in a completely different direction by building online password guessing directly into the tool, from which dictionaries generated with this tool will be used to guess passwords in runtime environments exactly mirroring current attacks underway in life. This added feature gives more reach to assessing vulnerabilities of passwords and real-world practical applicability for cybersecurity testing.

Chapter 12

Conclusions

This thesis surveyed the current state of the art in password generation and guessing tools; it examined their vulnerabilities due to PII used in passwords and finally developed an improved tool that would counter such vulnerabilities. This is intended to fill the gap between traditional brute-force methods to guessing passwords and more revised techniques tailored for personally identifiable information.

One of the significant contributions this thesis has to offer is designing and implementing an entirely new password generation tool that draws on PII to derive more accurate, faster, and highly efficient password guesses. The tool was designed to rectify some of the deficiencies in prior tools like CUPP by being much more targeted in its generation technique. This involved building dictionaries of special digits and letter characters in addition to probabilistic models that refined the guessing process, resulting in a faster and more efficient tool for guessing passwords.

The tool's performance was thoroughly tested against the CUPP tool along a set of metrics: verification of the average number of passwords generated, success rate, rank consistency, and time efficiency. The evaluations were performed on a real-world dataset, the 2016 leaked database dataset, which provided an applicable testbed with the challenges associated with the tool. The result shows that the developed tool has better results than CUPP in the majority of tested metrics. Notably, the guessing tool's success rate was higher. The ranking is more consistent, and the number of generated passwords is fewer. It demonstrates how well and how quickly the tool works.

The database dataset showed an even more astonishing state of password security. Among the overall findings was that a large proportion included personal information, and therefore, most are vulnerable or at high risk of targeted attacks. This highlights the need to educate users about the risk of having personal information in their passwords and also calls for stricter password policies to prevent such practices. This also supports the idea that attackers are exploiting this vulnerability. The personal information-based tool easily makes adaptation to these cybersecurity defenses critical.

Though this research has hit milestones, it had its limitations. First, using only one dataset, the database, reduces the generalization of findings. The limitation in the size of the dictionaries and the probabilistic threshold was unavoidable because of the time and computational limits of the CUPP tool; however, such choices limited the scope of analysis. Further, excluding some complex CUPP capabilities like leet speak and changing symbols does not make the test comprehensive. These limitations pointed to further research to broaden this tool and refine it with potential extension to a more extensive range of datasets or structures of passwords.

Despite all these limitations, this thesis clearly shows how more advanced tools can be developed to protect against the increasing threat of targeted password attacks.

Bibliography

[1] Abhishek, K., Roshan, S., Kumar, P., Ranjan, R.: A comprehensive study on multifactor authentication schemes. In: International Conference on Advances in Computing and Information Technology (2012) [Cited on pages 4 and 5.]

[2] Alshahrani, H., Alghamdi, A.: The factors influencing the use of password managers. Journal of Information Security and Cybercrimes Research 5, 43–56 (06 2022) [Cited on page 7.]

[3] Beyond Identity: Rainbow table attack. https://www.beyondidentity.com/glossary/rainbow-table-attack, accessed: 2024-07-10 [Cited on pages III and 18.]

[4] Blocki, J., Zhang, W.: Dalock: Password distribution-aware throttling. Proceedings on Privacy Enhancing Technologies 2022(3), 516–537 (2022) [Cited on page 36.]

[5] Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X.: The tangled web of password reuse (01 2014) [Cited on page 7.]

[6] Dong, Q., Wang, D., Shen, Y., Jia, C.: PII-PSM: A New Targeted Password Strength Meter Using Personally Identifiable Information, pp. 648–669 (02 2023) [Cited on page 49.]

[7] GeeksforGeeks: Understanding rainbow table attack. https://www.geeksforgeeks.org/understanding-rainbow-table-attack/ (February 10 2023), accessed: 2024-07-10 [Cited on pages III and 18.]

[8] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. Advances in Neural Information Processing Systems 3 (06 2014) [Cited on page 24.]

[9] Guo, Y., Zhang, Z., Guo, Y., Guo, X.: Nudging personalized password policies by understanding users' personality. Computers Security 94, 101801 (2020) [Cited on page 33.]

[10] Have I Been Pwned: Pwned websites. https://haveibeenpwned.com/PwnedWebsites, accessed: 2024-07-10 [Cited on page 39.]

[11] Karrar, A., Almutiri, T., Algrafi, S., Alalwi, N., Alharbi, A.: Enhancing salted password hashing technique using swapping elements in an array algorithm 9, 21–25 (01 2018) [Cited on page 34.]

[12] Kelseyk: Hybrid password attacks: How they work and how to stop them. https://specopssoft.com/blog/hybrid-password-attacks/ (August 31 2023), accessed: 2024-07-10 [Cited on pages III and 17.]

[13] Komanduri, S., Shay, R., Kelley, P.G., Mazurek, M.L., Bauer, L., Christin, N., Cranor, L.F., Egelman, S.: Of passwords and people: measuring the effect of password-composition policies. In: Tan, D.S., Amershi, S., Begole, B., Kellogg, W.A., Tungare, M. (eds.) Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011. pp. 2595–2604. ACM (2011) [Cited on page 11.]

[14] Li, Y., Wang, H., Sun, K.: A study of personal information in human-chosen passwords and its security implications. In: IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications. pp. 1–9 (2016) [Cited on page 41.]

[15] Liu, Y., Squires, M.R., Taylor, C.R., Walls, R.J., Shue, C.A.: Account lockouts: Characterizing and preventing account denial-of-service attacks. In: Chen, S., Choo, K.K.R., Fu, X., Lou, W., Mohaisen, A. (eds.) Security and Privacy in Communication Networks. pp. 26–46. Springer International Publishing, Cham (2019) [Cited on page 35.]

[16] Melicher, W., Ur, B., Segreti, S.M., Komanduri, S., Bauer, L., Christin, N., Cranor, L.F.: Fast, lean, and accurate: Modeling password guessability using neural networks. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 175–191. USENIX Association, Austin, TX (Aug 2016) [Cited on page 24.]

[17] Nita, S., Mihailescu, M., Pau, V.: Security and cryptographic challenges for authentication based on biometrics data. Cryptography 2, 39 (2018) [Cited on page 5.]

[18] NordVPN: Hybrid attack definition – glossary. `https://nordvpn.com/fr/cybersecurity/glossary/hybrid-attack/` (June 10 2024), accessed: 2024-07-10 [Cited on pages III and 17.]

[19] Pagnin, E., Mitrokotsa, A.: Privacy-preserving biometric authentication: Challenges and directions. Security and Communication Networks 2017(1), 7129505 (2017) [Cited on page 5.]

[20] Papathanasaki, M., Maglaras, L., Ayres, N.: Modern authentication methods: A comprehensive survey. AI, Computer Science and Robotics Technology 2022, 1–24 (06 2022) [Cited on pages 4, 7, 36, and 37.]

[21] Paul A. Grassi, M.E.G., Fenton, J.L.: Digital identity guidelines: Authentication and lifecycle management. National Institute of Standards and Technology (2017), `https://pages.nist.gov/800-63-3/sp800-63b.html`, accessed: 2024-07-21 [Cited on pages 4 and 8.]

[22] Raza, M., Iqbal, M., Sharif, M., Haider, W.: A survey of password attacks and comparative analysis on methods for secure authentication. World Applied Sciences Journal 19, 439–444 (01 2012) [Cited on pages 16, 17, 18, and 19.]

[23] Redmiles, E., Kross, S., Mazurek, M.: How i learned to be secure: a census-representative survey of security advice sources and behavior. pp. 666–677 (10 2016) [Cited on page 7.]

[24] Sahin, S., Roomi, S.A., Poteat, T., Li, F.: Investigating the password policy practices of website administrators. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 552–569 (2023) [Cited on pages 9 and 33.]

[25] Shay, R., Komanduri, S., Durity, A.L., Huh, P.S., Mazurek, M.L., Segreti, S.M., Ur, B., Bauer, L., Christin, N., Cranor, L.F.: Designing password policies for strength and usability. ACM Trans. Inf. Syst. Secur. 18(4) (may 2016) [Cited on page 33.]

[26] Singla, D., Verma, N.: Performance analysis of authentication system: A systematic literature review (01 2023) [Cited on pages 4, 5, 36, and 37.]

[27] Sutriman, Sugiantoro, B.: Analysis of password and salt combination scheme to improve hash algorithm security. International Journal of Advanced Computer Science and Applications 10(11) (2019) [Cited on page 34.]

[28] T. Ebanesar, G.S.: Improving login process by salted hashing password using sha-256 algorithm in web applications. International Journal of Computer Sciences and Engineering 7, 27–32 (3 2019) [Cited on page 34.]

[29] Tam, L., Glassman, M., Vandenwauver, M.: The psychology of password management: A tradeoff between security and convenience. Behaviour IT 29, 233–244 (05 2010) [Cited on pages 10 and 11.]

[30] Tan, J., Bauer, L., Christin, N., Cranor, L.F.: Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. p. 1407–1426. CCS '20, Association for Computing Machinery, New York, NY, USA (2020) [Cited on page 33.]

[31] Tran, L., Nguyen, T., Seo, C., Kim, H., Choi, D.: A survey on password guessing. ArXiv abs/2212.08796 (2022) [Cited on pages III, 14, 15, 23, and 24.]

[32] Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: An underestimated threat. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. p. 1242–1254. CCS '16, Association for Computing Machinery, New York, NY, USA (2016) [Cited on pages 14, 39, 43, 46, 47, 48, and 49.]

[33] Wash, R., Rader, E.: Prioritizing security over usability: Strategies for how people choose passwords. Journal of Cybersecurity 7(1), tyab012 (06 2021) [Cited on pages 10 and 11.]

[34] Weir, M., Aggarwal, S., de Medeiros, B., Glodek, B.: Password cracking using probabilistic context-free grammars. pp. 391–405 (05 2009) [Cited on pages 16, 17, 21, 41, 43, and 49.]

[35] Yu, F., Martin, M.V.: Gnpassgan: Improved generative adversarial networks for trawling offline password guessing. In: 2022 IEEE European Symposium on Security and Privacy Workshops (EuroSPW). pp. 10–18 (2022) [Cited on pages 14 and 15.]

Appendix A

Source code

The source code can be found at `https://gitlab.com/AliTazribine/password-generator`. Contact at ali.tazribine@ulb.be or alitazribine@gmail.com to get access

Appendix B

Documentation

The documentation can be found in the docs directory

Appendix C
Experimental data

Although all datasets used in this thesis could easily be accessed publicly, the passwords are sensitive data. We paid a lot of attention when manipulating the datasets. We reported aggregated statistics in this work—total number of passwords, common patterns, PII types distribution, and distribution of characters- to ensure none of the details of the individual accounts leaked out. Furthermore, to ensure privacy protection to a larger extent, secure offline systems were used for all data processing; sensitive information was irreversibly deleted once our analysis was over.